

CA-ADS[®]

Reference
15.0



Computer Associates™

This documentation and related computer software program (hereinafter referred to as the "Documentation") is for the end user's informational purposes only and is subject to change or withdrawal by Computer Associates International, Inc. ("CA") at any time.

This documentation may not be copied, transferred, reproduced, disclosed or duplicated, in whole or in part, without the prior written consent of CA. This documentation is proprietary information of CA and protected by the copyright laws of the United States and international treaties.

Notwithstanding the foregoing, licensed users may print a reasonable number of copies of this documentation for their own internal use, provided that all CA copyright notices and legends are affixed to each reproduced copy. Only authorized employees, consultants, or agents of the user who are bound by the confidentiality provisions of the license for the software are permitted to have access to such copies.

This right to print copies is limited to the period during which the license for the product remains in full force and effect. Should the license terminate for any reason, it shall be the user's responsibility to return to CA the reproduced copies or to certify to CA that same have been destroyed.

To the extent permitted by applicable law, CA provides this documentation "as is" without warranty of any kind, including without limitation, any implied warranties of merchantability, fitness for a particular purpose or noninfringement. In no event will CA be liable to the end user or any third party for any loss or damage, direct or indirect, from the use of this documentation, including without limitation, lost profits, business interruption, goodwill, or lost data, even if CA is expressly advised of such loss or damage.

The use of any product referenced in this documentation and this documentation is governed by the end user's applicable license agreement.

The manufacturer of this documentation is Computer Associates International, Inc.

Provided with "Restricted Rights" as set forth in 48 C.F.R. Section 12.212, 48 C.F.R. Sections 52.227-19(c)(1) and (2) or DFARS Section 252.227-7013(c)(1)(ii) or applicable successor provisions.

Second Edition, October 2001

© 2001 Computer Associates International, Inc.
All rights reserved.

All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Contents

How to Use This Manual	xiii
------------------------	------

Volume 1. CA-ADS Reference

Chapter 1. Introduction to CA-ADS	1-1
1.1 What is CA-ADS?	1-3
1.2 What CA-ADS does	1-4
1.3 Creating a CA-ADS application	1-5
1.4 Tools used to develop an application	1-7
1.4.1 The CA-ADS application compiler (ADSA)	1-7
1.4.2 Mapping facilities (MAPC and the Batch Compiler/Utility)	1-9
1.4.3 CA-ADS dialog compilers (ADSC and ADSOBCOM)	1-10
1.4.4 IDD menu facility and online IDD	1-12
1.4.5 The CA-ADS runtime system	1-13
1.5 CA-ADS screens	1-14
1.5.1 Action bar	1-15
1.5.2 Action bar actions	1-17
1.6 Checkout and release procedures	1-28
1.6.1 How to check out or release an entity	1-28
1.6.2 Listing checkouts (ADSL)	1-30
1.6.3 Modifying checkouts (ADSM)	1-31
1.7 CA-ADS help facility	1-32
 Chapter 2. CA-ADS Application Compiler (ADSA)	2-1
2.1 Overview	2-3
2.2 Application compiler session	2-4
2.2.1 Invoking the application compiler	2-4
2.2.2 Sequencing through application compiler screens	2-7
2.2.3 Suspending a session	2-10
2.2.4 Terminating a session	2-10
2.3 Application compiler screens	2-11
2.3.1 Main menu	2-11
2.3.2 General Options screen—Page 1	2-14
2.3.3 General Options screen—Page 2	2-16
2.3.4 Response/Function List screen	2-19
2.3.5 Response Definition screen	2-23
2.3.6 Function Definition (Dialog) screen	2-27
2.3.7 Function Definition (Program) screen	2-30
2.3.8 Function Definition (Menu) screen	2-32
2.3.9 Global Records screen	2-37
2.3.10 Task Codes screen	2-39
 Chapter 3. CA-ADS Dialog Compiler (ADSC)	3-1
3.1 Overview	3-3
3.2 Dialog compiler session	3-4
3.2.1 Invoking the dialog compiler	3-4

3.2.2	Sequencing through dialog compiler screens	3-5
3.2.3	Suspending a session	3-8
3.2.4	Terminating a session	3-9
3.3	Dialog compiler screens	3-10
3.3.1	Main menu	3-10
3.3.2	Options and Directives screen	3-13
3.3.3	Map Specifications screen	3-17
3.3.4	Database Specifications screen	3-20
3.3.5	Records and Tables screen	3-23
3.3.6	Process Modules screen	3-25
Chapter 4.	CA-ADS Runtime System	4-1
4.1	Initiating the CA-ADS runtime system	4-3
4.1.1	How to define runtime tasks	4-3
4.1.2	How to start a CA-ADS application	4-4
4.2	Runtime menu and help screens	4-8
4.2.1	Menu screens	4-8
4.2.2	Site-defined menu maps	4-9
4.2.3	System-defined menu maps	4-10
4.2.4	Application help screen	4-16
4.3	Runtime flow of control	4-19
4.3.1	Effects of automatic editing on flow of control	4-22
4.4	Message prefixes	4-23
4.5	CA-ADS tasks, run units, and transactions	4-24
4.5.1	Run units and database access	4-25
4.5.2	Extended run units	4-25
4.6	Dialog Abort Information screen	4-28
4.7	Debugging a dialog	4-32
4.8	Linking From CA-ADS to CA-OLQ	4-33
4.8.1	Linking to CA-OLQ	4-33
4.8.2	Passing syntax to CA-OLQ	4-33
4.9	Linking built-in functions with the runtime system	4-34
4.9.1	Linking system-supplied built-in functions	4-34
4.9.2	Linking user-written built-in functions	4-39
4.10	Managing storage	4-40
4.10.1	Adjusting record compression	4-40
4.10.2	Calculating RBB storage	4-40
4.10.3	Writing resources to scratch records	4-41
4.10.4	Using XA storage	4-42
Chapter 5.	Introduction to Process Language	5-1
5.1	Overview	5-3
5.2	Process modules	5-5
5.2.1	Creating process modules	5-5
5.2.2	Adding process modules to dialogs	5-5
5.2.3	Executing process modules	5-5
5.3	Process commands	5-7
5.3.1	Constructing commands	5-7
5.3.2	Coding considerations	5-8
5.4	Data types	5-10
5.4.1	Conversion between data types	5-16

Chapter 6. Arithmetic Expressions	6-1
6.1 Overview	6-3
6.2 Syntax	6-4
6.3 Evaluation of arithmetic expressions	6-5
6.4 Coding considerations	6-6
 Chapter 7. Built-in Functions	 7-1
7.1 Overview	7-3
7.1.1 Invocation names	7-3
7.1.2 Built-in function values	7-4
7.1.3 Coding parameters	7-4
7.2 User-defined built-in functions	7-5
7.3 System-supplied functions	7-6
7.3.1 Arithmetic functions	7-6
7.3.2 Date functions	7-6
7.3.3 String functions	7-7
7.3.4 Trailing-sign functions	7-8
7.3.5 Trigonometric functions	7-9
7.4 ABSOLUTE-VALUE	7-11
7.5 ARC COSINE	7-12
7.6 ARC SINE	7-13
7.7 ARC TANGENT	7-14
7.8 CONCATENATE	7-15
7.9 COSINE	7-16
7.10 DATECHG	7-17
7.11 DATEDIF	7-20
7.12 DATEOFF	7-21
7.13 EXTRACT	7-23
7.14 FIX	7-24
7.15 GOODDATE	7-25
7.16 GOODTRAILING	7-26
7.17 INITCAP	7-27
7.18 INSERT	7-28
7.19 INVERT-SIGN	7-30
7.20 LEFT-JUSTIFY	7-31
7.21 LIKE	7-32
7.22 LOGARITHM	7-34
7.23 MODULO	7-35
7.24 NEXT-INT-EQHI	7-36
7.25 NEXT-INT-EQLO	7-37
7.26 NUMERIC	7-38
7.27 RANDOM-NUMBER	7-40
7.28 REPLACE	7-42
7.29 RIGHT-JUSTIFY	7-44
7.30 SIGN-VALUE	7-45
7.31 SINE	7-46
7.32 SQUARE-ROOT	7-47
7.33 STRING-INDEX	7-48
7.34 STRING-LENGTH	7-49
7.35 STRING-REPEAT	7-50

7.36	SUBSTRING	7-51
7.37	TANGENT	7-53
7.38	TODAY	7-54
7.39	TOLOWER	7-55
7.40	TOMORROW	7-56
7.41	TOUPPER	7-57
7.42	TRAILING-TO-ZONED	7-58
7.43	TRANSLATE	7-59
7.44	VERIFY	7-61
7.45	WEEKDAY	7-62
7.46	WORDCAP	7-65
7.47	YESTERDAY	7-66
7.48	ZONED-TO-TRAILING	7-67
Chapter 8. Conditional Expressions		8-1
8.1	Overview	8-3
8.2	General considerations	8-4
8.2.1	Syntax for conditional expressions	8-4
8.3	Batch-control event condition	8-6
8.4	Command status condition	8-7
8.5	Comparison condition	8-10
8.6	Cursor position condition	8-12
8.7	Dialog execution status condition	8-14
8.8	Environment status condition	8-16
8.9	Level-88 condition	8-17
8.10	Map field status condition	8-18
8.11	Map paging status conditions	8-22
8.12	Set status condition	8-25
8.13	Arithmetic and assignment command status condition	8-27
Chapter 9. Constants		9-1
9.1	Overview	9-3
9.2	Figurative constants	9-4
9.3	Graphic literals	9-6
9.4	Multibit binary constants	9-7
9.5	Nonnumeric literals	9-8
9.6	Numeric literals	9-9
Chapter 10. Error Handling		10-1
10.1	Overview	10-3
10.2	The autostatus facility	10-4
10.3	Error expressions	10-6
10.4	The ALLOWING clause	10-7
10.5	Status definition records	10-9
Chapter 11. Variable Data Fields		11-1
11.1	Overview	11-3
11.2	User-defined data field names	11-4
11.3	System-supplied data field names	11-6
11.4	Entity names	11-12

Chapter 12. Introduction to Process Commands	12-1
12.1 Overview	12-3
12.2 Summary of process commands	12-4
12.3 INCLUDE	12-8
 Chapter 13. Arithmetic and Assignment Commands	 13-1
13.1 Overview	13-3
13.2 General considerations	13-4
13.2.1 Numeric fields	13-4
13.2.2 EBCDIC and DBCS fields	13-4
13.2.3 Arithmetic and assignment command status condition	13-5
13.3 Arithmetic commands	13-6
13.3.1 ADD	13-6
13.3.2 COMPUTE	13-7
13.3.3 DIVIDE	13-8
13.3.4 MULTIPLY	13-10
13.3.5 SUBTRACT	13-11
13.4 Assignment command	13-13
13.4.1 MOVE	13-14
 Chapter 14. Conditional Commands	 14-1
14.1 Overview	14-3
14.2 EXIT	14-4
14.3 IF	14-5
14.4 NEXT	14-8
14.5 WHILE	14-10

Volume 2. CA-ADS Reference

Chapter 15. Control Commands	15-1
15.1 Overview	15-3
15.2 General considerations	15-5
15.2.1 Application thread	15-5
15.2.2 Operative and nonoperative dialogs	15-6
15.2.3 Application levels	15-6
15.2.4 Mainline dialog	15-6
15.2.5 The menu stack	15-7
15.2.6 Database currencies	15-7
15.3 CONTINUE	15-10
15.4 DISPLAY	15-12
15.5 EXECUTE NEXT FUNCTION	15-17
15.6 INVOKE	15-19
15.7 LEAVE	15-22
15.8 LINK	15-24
15.9 READ TRANSACTION	15-30
15.10 RETURN	15-31
15.11 TRANSFER	15-34
15.12 WRITE TRANSACTION	15-36

Chapter 16. Database Access Commands	16-1
16.1 Overview	16-3
16.2 Navigational DML	16-5
16.2.1 Overview of navigational database access	16-5
16.2.2 Use of native VSAM data sets	16-7
16.2.3 Record locking	16-9
16.2.4 Suppression of record retrieval locks	16-10
16.2.5 Overview of ACCEPT	16-12
16.2.6 ACCEPT DB-KEY FROM CURRENCY	16-12
16.2.7 ACCEPT DB-KEY RELATIVE TO CURRENCY	16-14
16.2.8 ACCEPT PAGE-INFO	16-16
16.2.9 ACCEPT STATISTICS	16-17
16.2.10 BIND PROCEDURE	16-19
16.2.11 COMMIT	16-20
16.2.12 CONNECT	16-22
16.2.13 DISCONNECT	16-25
16.2.14 ERASE	16-27
16.2.15 Overview of FIND/OBTAIN	16-30
16.2.16 FIND/OBTAIN CALC	16-31
16.2.17 FIND/OBTAIN CURRENT	16-33
16.2.18 FIND/OBTAIN DB-KEY	16-34
16.2.19 FIND/OBTAIN OWNER	16-37
16.2.20 FIND/OBTAIN WITHIN SET/AREA	16-38
16.2.21 FIND/OBTAIN WITHIN SET USING SORT KEY	16-42
16.2.22 GET	16-44
16.2.23 KEEP	16-46
16.2.24 KEEP LONGTERM	16-47
16.2.25 MODIFY	16-53
16.2.26 READY	16-55
16.2.27 RETURN DB-KEY	16-57
16.2.28 ROLLBACK	16-59
16.2.29 STORE	16-60
16.3 Logical Record Facility commands	16-64
16.3.1 Overview of LRF database access	16-64
16.3.2 WHERE clause	16-65
16.3.3 Conditional expression	16-65
16.3.4 Comparison expression	16-66
16.3.5 ERASE	16-68
16.3.6 MODIFY	16-69
16.3.7 OBTAIN	16-70
16.3.8 ON command	16-71
16.3.9 STORE	16-75
 Chapter 17. Map Commands	 17-1
17.1 Overview	17-3
17.2 Map modification commands	17-4
17.3 Attributes Command	17-5
17.4 CLOSE	17-10
17.5 MODIFY MAP	17-12
17.6 Pageable maps	17-21
17.6.1 Areas of a pageable map	17-21

17.6.2	Map paging session	17-22
17.6.3	Map paging dialog options	17-27
17.6.4	GET DETAIL	17-28
17.6.5	PUT DETAIL	17-30
17.6.6	Creating or modifying a detail occurrence of a pageable map	17-32
17.6.7	Specifying a numeric value associated with an occurrence	17-32
17.6.8	Specifying a message to appear in the message field of an occurrence	17-32
Chapter 18. Queue and Scratch Management Commands		18-1
18.1	Overview	18-3
18.2	Queue records	18-5
18.3	DELETE QUEUE	18-7
18.4	GET QUEUE	18-9
18.5	PUT QUEUE	18-12
18.6	Scratch records	18-15
18.6.1	CA-ADS usage	18-15
18.6.2	CA-ADS/Batch considerations	18-16
18.7	DELETE SCRATCH	18-17
18.8	GET SCRATCH	18-19
18.9	PUT SCRATCH	18-22
Chapter 19. Subroutine Control Commands		19-1
19.1	Overview	19-3
19.2	CALL	19-4
19.3	DEFINE	19-5
19.4	GOBACK	19-6
Chapter 20. Utility Commands		20-1
20.1	Overview	20-3
20.2	ABORT	20-4
20.3	ACCEPT	20-8
20.4	INITIALIZE RECORDS	20-10
20.5	SNAP	20-11
20.6	TRACE	20-13
20.7	WRITE PRINTER	20-14
20.8	WRITE TO LOG/OPERATOR	20-18
Chapter 21. Cooperative Processing Commands		21-1
21.1	Using SEND/RECEIVE commands	21-3
21.1.1	How cooperative processing works	21-3
21.2	Sample cooperative application	21-4
21.2.1	Program A: Client listing (PC)	21-5
21.2.2	Dialog B: Server listing (Mainframe)	21-7
21.3	SEND/RECEIVE commands	21-9
21.4	ALLOCATE	21-10
21.5	CONFIRM	21-13
21.6	CONFIRMED	21-14
21.7	CONTROL SESSION	21-15
21.8	DEALLOCATE	21-17

21.9	PREPARE-TO-RECEIVE	21-19
21.10	RECEIVE-AND-WAIT	21-20
21.11	REQUEST-TO-SEND	21-21
21.12	SEND-DATA	21-22
21.13	SEND-ERROR	21-24
21.14	Design guidelines	21-25
21.15	Understanding conversation states	21-26
21.15.1	Conversation states in a successful data transfer	21-28
21.16	Testing APPC status codes and system fields	21-30
21.16.1	Status codes	21-30
21.16.2	System fields	21-30
21.16.3	When APPC status codes and system field values are returned	21-30
21.16.4	APPCCODE and APPCERC	21-31
21.16.5	System fields	21-34
Chapter 22.	OSCaR Commands	22-1
22.1	OSCaR command syntax	22-4
22.1.1	OPEN	22-4
22.1.2	SEND	22-5
22.1.3	CLOSE	22-6
22.1.4	RECEIVE	22-6
22.2	Sample OSCaR application	22-7
22.3	OSCaR to APPC Mapping	22-9
Appendix A.	System Records	A-1
A.1	Overview	A-3
A.2	ADSO-APPLICATION-GLOBAL-RECORD	A-4
A.3	ADSO-APPLICATION-MENU-RECORD	A-15
Appendix B.	CA-ADS Dialog and Application Reporter	B-1
B.1	Overview	B-3
B.2	Dialog reports	B-4
B.3	Application reports	B-15
B.4	Control statements	B-16
B.4.1	APPLICATIONS	B-16
B.4.2	DIALOGS	B-18
B.4.3	LIST	B-21
B.4.4	SEARCH	B-22
B.5	SYSIDMS parameter file	B-24
B.6	JCL and commands to run reports	B-25
Appendix C.	Dialog Statistics	C-1
C.1	Overview	C-3
C.2	Collecting selected statistics	C-4
C.3	Enabling dialog statistics	C-8
C.4	Selecting dialogs	C-9
C.5	Setting a checkpoint interval	C-10
C.6	Collecting and writing statistics	C-11
C.7	Statistics reporting	C-12
Appendix D.	Application and Dialog Utilities	D-1

D.1	Overview	D-3
D.2	ADSOBCOM	D-4
D.2.1	Standard control statements	D-4
D.2.2	Special control statements	D-5
D.2.3	SIGNON	D-5
D.2.4	COMPILE	D-6
D.2.5	DECOMPILE	D-8
D.2.6	Dialog-expression	D-10
D.2.7	JCL and commands	D-30
D.2.7.1	OS/390 JCL	D-30
D.2.7.2	VSE/ESA JCL	D-31
D.2.7.3	VM/ESA commands	D-33
D.2.7.4	BS2000/OSD JCL	D-35
D.3	ADSOBSYS	D-37
D.3.1	Control statements	D-37
D.3.2	SYSTEM statement	D-38
D.3.3	JCL and commands	D-39
D.3.3.1	OS/390 JCL	D-39
D.3.3.2	VSE/ESA JCL	D-42
D.3.3.3	VM/ESA commands	D-44
D.3.3.4	BS2000/OSD JCL	D-46
D.4	ADSOBTAT	D-48
D.4.1	Control statements	D-49
D.4.2	JCL and commands	D-51
D.4.2.1	OS/390 JCL	D-51
D.4.2.2	VSE/ESA JCL	D-52
D.4.2.3	VM/ESA commands	D-54
D.4.2.4	BS2000/OSD JCL	D-55
D.5	ADSOTATU	D-57
D.5.1	TAT update utility screen	D-58
Appendix E. Activity Logging for a CA-ADS Dialog		E-1
E.1	Overview	E-3
E.2	Data dictionary organization	E-4
E.3	Activity logging record formats	E-5
Appendix F. Built-in Function Support		F-1
F.1	Overview	F-3
F.2	Internal structure of built-in functions	F-4
F.2.1	Master function table	F-5
F.2.2	Model XDE module	F-6
F.2.3	XDEs and VXDEs	F-8
F.2.4	Processing program modules	F-17
F.2.5	Runtime processing of built-in functions	F-24
F.3	Assembler macros	F-27
F.3.1	#EFUNMST	F-27
F.3.2	RHDCEVBF	F-28
F.3.3	#EFUNMOD	F-31
F.4	Changing invocation names	F-40
F.5	Creating user-defined built-in functions	F-41

F.5.1	Steps for generating a user-defined built-in function	F-41
F.5.2	LRF considerations for user-defined built-in functions	F-42
F.5.3	Calling a user-defined built-in function	F-42
Appendix G. Security Features		G-1
G.1	Overview	G-3
G.2	CA-ADS compiler security	G-4
G.3	CA-ADS application security	G-5
G.3.1	Response security	G-5
G.3.2	Signon security	G-6
Appendix H. Debugging a CA-ADS Dialog		H-1
H.1	Creating a symbol table	H-4
H.2	Trace facility	H-5
H.3	Online debugger	H-7
Index		X-1

How to Use This Manual

What this manual is about

This manual is a reference for Application Development System (CA-ADS®) development tools and facilities. It provides reference information appropriate for application developers defining online and batch applications.

The manual is divided into two volumes.

Volume 1:

- Introduces CA-ADS and provides information about tools used to develop applications
- Introduces the process language and error-handling facility used in developing CA-ADS applications

Volume 2:

- Presents syntax and examples of process language command statements used to construct processing routines
- Presents detailed information about facilities and utilities available to the CA-ADS application developer

New users may find it helpful to familiarize themselves with the *CA-ADS User Guide* before relying solely on this reference manual. Experienced users can use this reference as needed. Developers using CA-ADS/Batch® extensions to CA-ADS to define batch applications should refer to *CA-ADS Batch User Guide*. Syntax for developing batch applications is included in this reference manual.

Related documentation

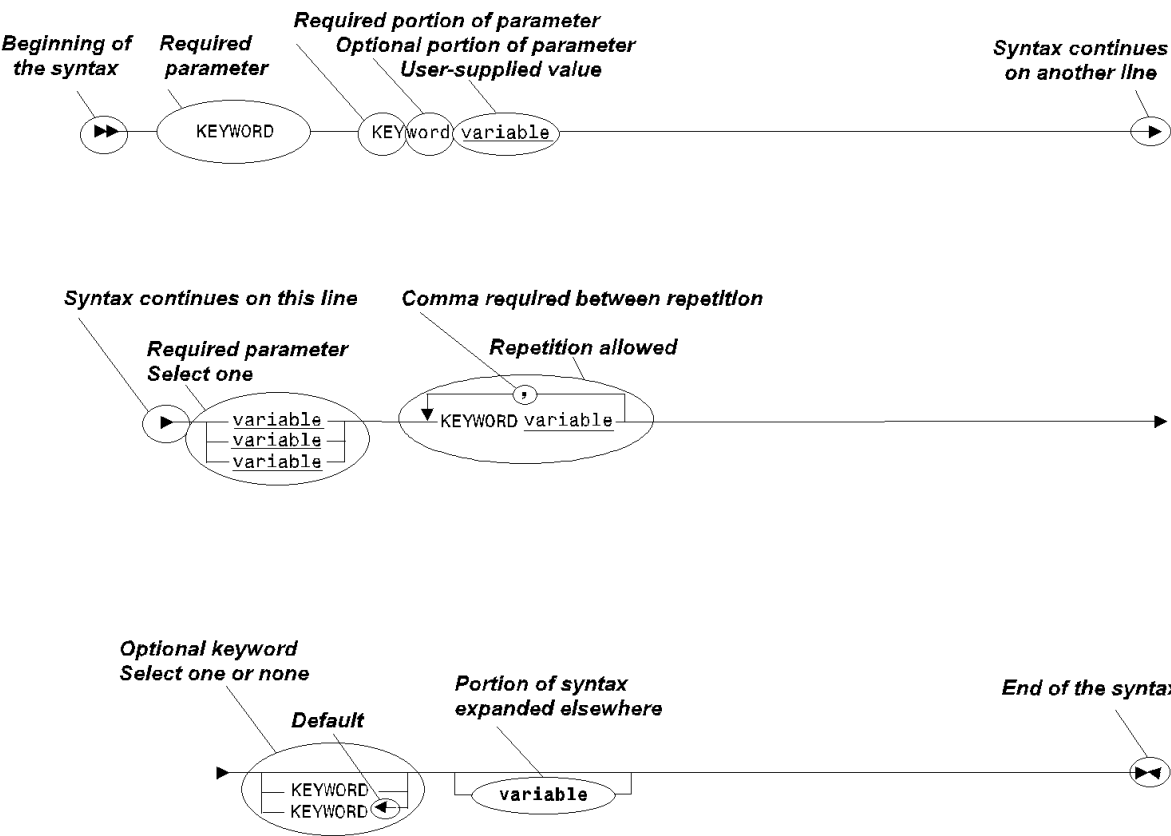
- *CA-ADS Quick Reference*
- *CA-ADS User Guide*
- *CA-IDMS Mapping Facility*
- *CA-IDMS Mapping Facility quick Reference*
- *IDD DDDL Reference*
- *CA-ADS DSECT Reference*
- *CA-IDMS SQL Programming*
- *CA-ADS Batch User Guide*
- *CA-IDMS Security Administration*
- *CA-IDMS System Generation*

Understanding Syntax Diagrams

Look at the list of notation conventions below to see how syntax is presented in this manual. The example following the list shows how the conventions are used.

UPPERCASE OR SPECIAL CHARACTERS	Represents a required keyword, partial keyword, character, or symbol that must be entered completely as shown.
lowercase	Represents an optional keyword or partial keyword that, if used, must be entered completely as shown.
<u>underlined lowercase</u>	Represents a value that you supply.
←	Points to the default in a list of choices.
lowercase bold	Represents a portion of the syntax shown in greater detail at the end of the syntax or elsewhere in the document.
▶▶	Shows the beginning of a complete piece of syntax.
◀◀	Shows the end of a complete piece of syntax.
▶	Shows that the syntax continues on the next line.
▶	Shows that the syntax continues on this line.
▶	Shows that the parameter continues on the next line.
▶	Shows that a parameter continues on this line.
▶ parameter ▶	Shows a required parameter.
▶ parameter parameter ▶	Shows a choice of required parameters. You must select one.
▶ parameter ▶	Shows an optional parameter.
▶ parameter parameter ▶	Shows a choice of optional parameters. Select one or none.
▶ parameter ▶	Shows that you can repeat the parameter or specify more than one parameter.
▶ parameter , parameter ▶	Shows that you must enter a comma between repetitions of the parameter.

Sample Syntax Diagram



Volume 2. CA-ADS Reference

Chapter 15. Control Commands

15.1 Overview	15-3
15.2 General considerations	15-5
15.2.1 Application thread	15-5
15.2.2 Operative and nonoperative dialogs	15-6
15.2.3 Application levels	15-6
15.2.4 Mainline dialog	15-6
15.2.5 The menu stack	15-7
15.2.6 Database currencies	15-7
15.3 CONTINUE	15-10
15.4 DISPLAY	15-12
15.5 EXECUTE NEXT FUNCTION	15-17
15.6 INVOKE	15-19
15.7 LEAVE	15-22
15.8 LINK	15-24
15.9 READ TRANSACTION	15-30
15.10 RETURN	15-31
15.11 TRANSFER	15-34
15.12 WRITE TRANSACTION	15-36

15.1 Overview

CA-ADS control commands are used to pass control during the execution of an application. The execution of a control command terminates the execution of the process that issues the command. A control command can pass control to:

- Another dialog
- A copy of the same dialog
- Another component within the same dialog
- A user-written program
- Another application function when using the EXECUTE NEXT FUNCTION command

Summary of control commands: Control commands are listed in the table below. Each command is presented in alphabetical order after 15.2, “General considerations.”

Command	Purpose
CONTINUE	Terminates the current process, executes the dialog's premap process, and writes a message
DISPLAY	Displays a dialog's map, reexecutes a dialog's premap process, or specifies a message that appears in a map's message field
EXECUTE NEXT FUNCTION	Passes control to the application function associated with a response by means of the control command specified for the response during application compilation
INVOKE	Initiates execution of a lower level dialog in the application thread
LEAVE	Terminates the current application or terminates the current CA-ADS session
LINK	Initiates execution of a lower level dialog, creating a nested application structure, or initiates execution of a user program
READ TRANSACTION	Terminates the current process, performs a mapin operation, and selects the next application function or response to be executed (batch only)
RETURN	Initiates execution of a higher level dialog
TRANSFER	Initiates execution of a dialog at the same level as the dialog passing control

Command	Purpose
WRITE TRANSACTION	Terminates a current process, performs a mapout operation, and passes control within an application (batch only)

15.2 General considerations

At run time, control commands connect the application functions or dialogs that make up the application by directing the flow of control. The diagram below shows how control commands pass control from one function or dialog to another. The way that control is transferred determines the data that is available to the function or dialog when it receives process control.

The application developer associates control commands with application responses by using the Response Definition screen during application compilation.

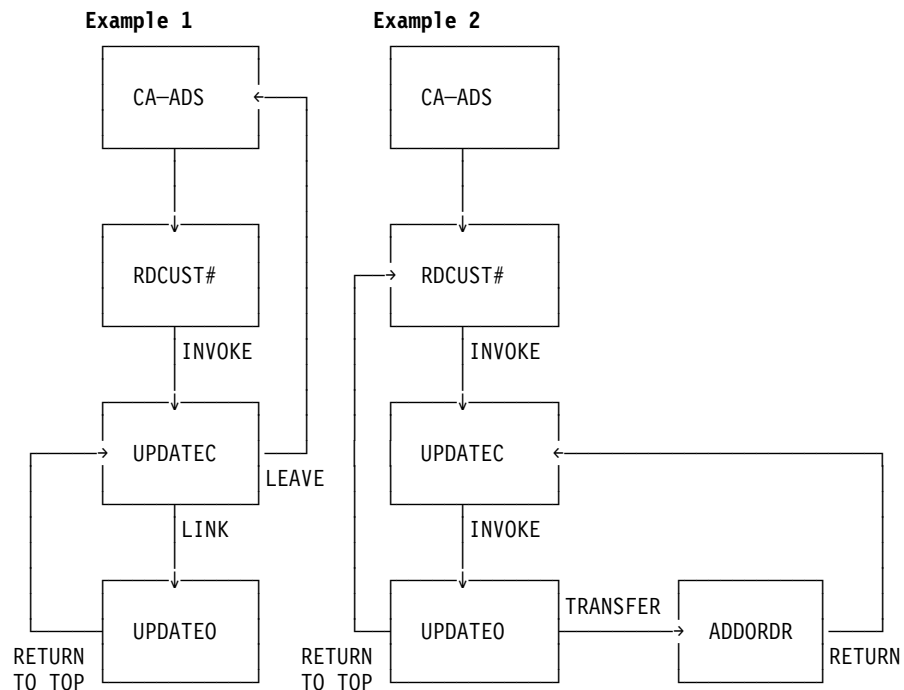
►► For more information on associating a control command with a response, see "Response Definition screen" in Chapter 2, "CA-ADS Application Compiler (ADSA)."

Alternatively, the application developer can code control commands wherever appropriate in a premap or response process.

15.2.1 Application thread

The current sequence of operative functions or dialogs in an application is called the application thread. A single dialog can occur more than once in an application structure and can execute more than once within an application thread, whether or not the function or dialog remains operative.

Control command processing



15.2.2 Operative and nonoperative dialogs

At run time, a function or dialog can be either operative or nonoperative within an application thread.

Operative dialog: A dialog becomes **operative** when it receives processing control. A function or dialog remains operative when it passes control to a lower level function or dialog or to another part of itself.

Only one dialog can be operative at any time on any given application level. As long as a dialog or dialog function remains operative, all record buffers associated with the dialog are maintained.

Nonoperative dialog: A function or dialog becomes **nonoperative** when it passes control to a higher level-function or dialog or to a function or dialog (including a copy of itself) on the same level. All functions and dialogs become nonoperative when control passes out of the application.

When a dialog or dialog function becomes nonoperative, the record buffers established by that dialog are released.

15.2.3 Application levels

The first function or dialog executed in an application establishes the top level of the application structure. The INVOKE and LINK commands establish lower levels in the structure.

Maximum number of levels: By default, an application structure can contain a maximum of ten levels. This maximum number of levels can be reduced at system generation time. If the execution of an INVOKE or LINK command causes the maximum allowable number of levels to be exceeded, CA-ADS abnormally terminates the application. The application developer should limit the total number of nested INVOKE and LINK commands accordingly.

15.2.4 Mainline dialog

The dialog at the top of an application structure must be a mainline dialog. The application developer defines a dialog as mainline by using the Options and Directives screen of the dialog compiler

►► The Options and Directives screen is described in Chapter 3, “CA-ADS Dialog Compiler (ADSC).”

If a dialog function is initiated by an application task code, the dialog associated with the function must be a mainline dialog. The application developer associates a function with a task code by using the Task Codes screen during application definition.

►► For a description of the Task Codes screen, see Chapter 2, “CA-ADS Application Compiler (ADSA).”

15.2.5 The menu stack

System-supplied menu-handling routines use a menu stack to keep track of menu execution at run time. The menu stack is maintained automatically at run time.

Considerations: The following considerations apply:

- When a menu (or menu/dialog) function is executed in an online application, the function name is added to the internal menu stack. The menu or menu/dialog function name is removed from the menu stack when a POP or RETURN function returns control in either of the following ways:
 - To the menu
 - To a menu that is higher in the menu stack
- If a menu or menu dialog function is already in the menu stack when a LINK, INVOKE, or TRANSFER command passes control again to the function, the first occurrence of the name is deleted from the stack. The name is then added to the end of the stack, as usual.
- Each menu name can appear only once in the menu stack.

15.2.6 Database currencies

Database currencies are established by the last database command in an operative dialog. Currencies are saved and made available to lower level dialogs and to the dialog that established the currencies if control returns to that dialog from a lower application level.

Considerations: The following considerations apply to currencies:

- Database currencies are cumulative.

Currencies established by each dialog or dialog function are passed to lower level dialogs along with any currencies received from a higher level dialog. A lower level dialog can establish new currencies, which are passed to the next lower level dialog along with the currencies already established.
- All database currencies established for a dialog are released when a dialog or a dialog function becomes nonoperative.

Unless the dialog or dialog function receiving control specifies the NOSAVE keyword on a LINK command, it establishes its own currencies. These currencies are established either by restoring the currencies saved when it originally passed control or by using currencies previously established by a higher level dialog. The diagram below shows currencies in a CA-ADS application.

Currency action

DIALOG A

Currencies			
Received		Established	
EMP	A	EMP	A
NONE		DEPT	X

DIALOG A executes and establishes current records of two sets.

DIALOG B

Currencies			
Received		Established	
EMP	A	EMP	A
DEPT	X	DEPT	X
		OFFICE	M

DIALOG A links to DIALOG B and establishes a current record of a third set.

DIALOG C

Currencies			
Received		Established	
EMP	B	EMP	C
DEPT	Y	DEPT	Z
OFFICE	M		

DIALOG B invokes DIALOG C.

DIALOG C returns control to DIALOG A:

- Only currencies established by DIALOG A are available.
- Record buffers still contain data established by DIALOG C

Effect of control commands on issuing and receiving dialogs: The following table outlines the effect of control commands on issuing and receiving dialogs. The EXECUTE NEXT FUNCTION command is not included in this table. The characteristics established by EXECUTE NEXT FUNCTION depend on which command is actually executed.

Command	New level established	Status of issuing dialog	Data avail. to receiving dialog/program	Currency action for issuing dialog	Currency action for receiving dialog/program
DISPLAY	No	Operative	All data	Saved	N/A
INVOKE	Yes	Operative	All data	Saved	Restored
LEAVE	No	Non-operative	N/A	Released	N/A
LINK:					

Command	New level established	Status of issuing dialog	Data avail. to receiving dialog/program	Currency action for issuing dialog	Currency action for receiving dialog/program
DIALOG	Yes	Operative	All data	Saved, unless NOSAVE is specified	Restored
PROGRAM	No	Operative	All, some, or none (depending on command specification)	Saved, unless NOSAVE is specified	Program receives currencies as part of extended run unit
RETURN	No	Non-operative (any operative dialogs between the issuing dialog and the receiving dialog also become non-operative)	Data previously available to the receiving dialog	Released (currencies for any dialogs between the issuing dialog and the receiving dialog are also released)	Restored
TRANSFER	No	Non-operative	All data except that acquired by the issuing dialog	Released	Can use currencies previously established by higher level dialogs

15.3 CONTINUE

Purpose: Terminates a current process, executes a dialog's premap process, and specifies a message.

In the online environment, the message appears at the terminal when the dialog executes a DISPLAY command. In the batch environment, the message is sent to the log file and/or the operator's console.

Syntax

```

>> CONTINUE [ MESSAge | MSG ] message-options .

```

Expansion of message-options

```

>> [ TEXT [ IS ] message-text ]
    [ CODE [ IS ] message-code ]

[ PARS [ = ] ( parameter ) ]

[ PREFIX [ IS ] prefix ]

```

Parameters

MESSAge message-options

Identifies message to be displayed.

MSG can be used in place of MESSAGE.

Expanded syntax for message-options is shown above immediately following the CONTINUE syntax.

TEXT IS message-text

Specifies the text of a message to be displayed in an online map's message field or sent to a batch application and a system log file.

Message-text specifies either the name of a variable data field containing the message text or the text string itself, enclosed in single quotation marks.

The text string can contain up to 240 displayable characters.

IS or = are optional keywords and have no effect on processing.

CODE IS message-code

Specifies the message dictionary code of a message to be displayed in an online map's message field or sent to the log file in a batch application.

In a batch application, the message is also sent to the operator, if directed by the destination specified in the dictionary.

Message-code specifies either the name of a variable data field that contains the message code or the 6-digit code itself, expressed as a numeric literal.

IS or = are optional keywords and have no effect on processing.

PARMS = parameter

Specifies a replacement parameter for each variable field in the stored message identified by *message-code*.

Up to nine replacement parameters can be specified for a message. Multiple parameters must be specified in the order in which they are numbered and separated by blanks or commas.

Parameter specifies either the name of an EBCDIC or unsigned zoned decimal variable data field that contains the parameter value or the actual parameter value, enclosed in single quotation marks.

The parameter value must contain displayable characters. At run time, each variable data field in a stored message expands or contracts to accommodate the size of its replacement parameter. A replacement parameter can be a maximum of 240 bytes.

PREFIX IS prefix

Overrides the default prefix of a dialog and a map.

Prefix must either specify an EBCDIC or unsigned zoned decimal variable data field that contains a 2-character prefix or the 2-character prefix itself, enclosed in single quotation marks

IS or = are optional keywords and have no effect on processing.

Usage*Considerations*

- The premap process is reexecuted if CONTINUE is issued in the premap process. This differs from the DISPLAY CONTINUE command, which causes a pseudo-converse in the online environment.
- Any message specified on the CONTINUE command is ignored in the online environment if the DISPLAY command that follows also specifies a message.
- Up to nine replacement parameters can be specified for a message.
- Multiple parameters must be separated by blanks or commas.
- Multiple parameters must be specified in the order in which they occur in the stored message.

15.4 DISPLAY

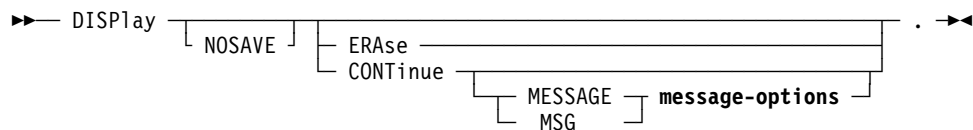
Purpose: Displays a dialog's map, or reexecutes a dialog's premap process.

Additionally, DISPLAY can specify a message that appears in a map's message field. If a dialog has a map and a premap process, the premap process must include a DISPLAY command to display the map. If a DISPLAY command is not coded, nothing is written to the terminal at run time.

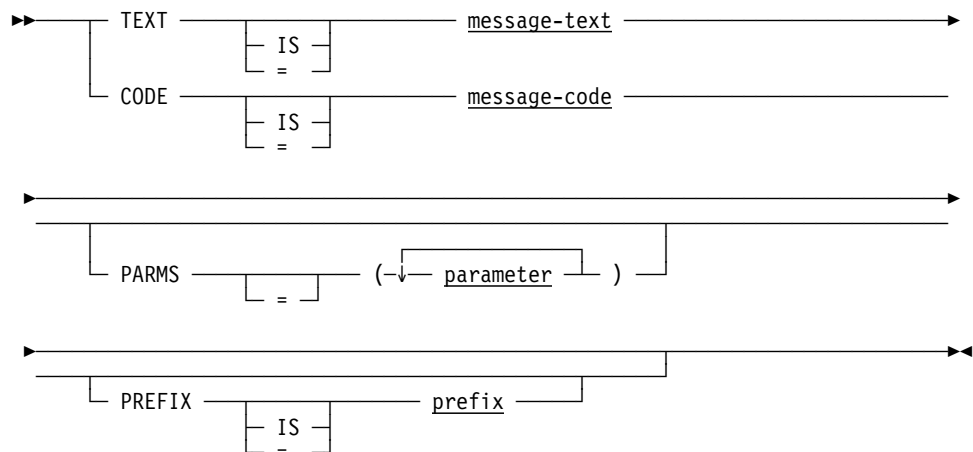
DISPLAY issued without the CONTINUE keyword, displays the map associated with the current dialog. DISPLAY can be used in a premap process or a response process.

In a pageable map, the detail occurrences that are displayed when the DISPLAY command is issued depend on the value of the system-defined data field \$PAGE and the number of detail occurrences that a single screen can hold. For example, given a screen that can hold ten detail lines, if \$PAGE equals 1, detail occurrences 1 through 10 are displayed; if \$PAGE equals 2, occurrences 11 through 20 are displayed; and so forth.

Syntax



Expansion of message-options



Parameters

NOSAVE

Specifies that currencies are not saved when control passes from the current process to the pseudo-converse or premap process. After the pseudo-converse, or when the premap process begins execution, the dialog's currencies are initialized to those of the next higher level dialog, if any.

ERase

Specifies that the following actions are performed at the terminal:

- Unprotected map data fields are cleared.
- The modified data tags (MDTs) for all unprotected map data fields are reset.
- The keyboard is unlocked.
- The cursor is placed at the first unprotected map data field.

If specified, ERASE is the only keyword that can follow DISPLAY in a DISPLAY command.

CONTinue

(Used in a response process) Requests reexecution of the premap process associated with the current dialog.

The keyword CONTINUE is ignored in a premap process.

MESSage message-options

Identifies message to be displayed.

MSG can be used in place of MESSAGE.

Expanded syntax for message-options is shown above immediately following the CONTINUE syntax.

TEXT IS message-text

Specifies the text of a message to be displayed in an online map's message field or sent to a batch application and a system log file.

Message-text specifies either the name of a variable data field containing the message text or the text string itself, enclosed in single quotation marks.

The text string can contain up to 240 displayable characters.

IS or = are optional keywords and have no effect on processing.

CODE IS message-code

Specifies the message dictionary code of a message to be displayed in an online map's message field or sent to the log file in a batch application.

In a batch application, the message is also sent to the operator, if directed by the destination specified in the dictionary.

Message-code specifies either the name of a variable data field that contains the message code or the 6-digit code itself, expressed as a numeric literal.

IS or = are optional keywords and have no effect on processing.

PARMS = parameter

Specifies a replacement parameter for each variable field in the stored message identified by *message-code*.

Up to nine replacement parameters can be specified for a message. Multiple parameters must be specified in the order in which they are numbered and separated by blanks or commas.

Parameter specifies either the name of an EBCDIC or unsigned zoned decimal variable data field that contains the parameter value or the actual parameter value, enclosed in single quotation marks.

The parameter value must contain displayable characters. At run time, each variable data field in a stored message expands or contracts to accommodate the size of its replacement parameter. A replacement parameter can be a maximum of 240 bytes.

PREFIX IS prefix

Overrides the default prefix of a dialog and a map.

Prefix must either specify an EBCDIC or unsigned zoned decimal variable data field that contains a 2-character prefix or the 2-character prefix itself, enclosed in single quotation marks

IS or = are optional keywords and have no effect on processing.

Usage

Rules for mapping out fields: The DISPLAY command maps out literal fields and data fields according to these rules:

- If the map is different than the map previously displayed, both literal fields and data fields are mapped out.
- If the map is the same as the map previously displayed, literal fields are not mapped out. Data fields, except those set IN ERROR, are mapped out. Note that the MODIFY MAP command can be used to change the IN ERROR setting for a map field.
- If the ERASE keyword is specified, data fields are not mapped out. Instead, unprotected data fields on the screen are cleared.
- Data fields are further regulated by specifications made during map definition and by MODIFY MAP process commands. Both methods allow the specification that data is not displayed or is erased on a DISPLAY command.
- For a pageable map, if a PUT DETAIL command causes the first map page to be displayed, the following DISPLAY command does not map out literal or data fields. However, the DISPLAY command is still required to terminate the current process and create a pseudo-converse.

Specifying a message: The DISPLAY command is also used to specify a message that is to appear in a map's message field.

Message fields are defined by the map field \$MESSAGE.

►► For more information, refer to *CA-IDMS Mapping Facility*.

One \$MESSAGE field can be defined anywhere on the map. If the \$MESSAGE field is defined in the detail area of a pageable map, the PUT DETAIL command is used to specify a message.

►► For more information, see "PUT DETAIL" in Chapter 17, "Map Commands."

If a DISPLAY command specifies a message but the map has no message field, CA-ADS creates a special message map.

Considerations for specifying a message code

- Each message in the message dictionary is identified by a 6-digit code preceded by the letters DC. A request for message 987654 retrieves message DC987654.
User-defined messages added to the message dictionary should be identified by a code in the range 900001 through 999999 and preceded by letters other than DC.
- Each message in the message dictionary can be assigned a severity code. The severity code specifies the action that CA-ADS takes when the message is retrieved. Severity codes are listed in the following table.

Message dictionary severity codes

Severity code	Action
0	Processes the DISPLAY command
1	Snaps all CA-ADS resources and processes the DISPLAY command
2	Snaps all system areas and processes the DISPLAY command
3	Snaps all CA-ADS resources and terminates CA-ADS with a task abend code of D002
4	Snaps all system areas and terminates CA-ADS with a task abend code of D002
5	Terminates CA-ADS with a task abend code of D002
8	Snaps all system areas and terminates the DC system with an operating system abend code of 3996
9	Terminates the DC system with an operating system abend code of 3996

A message in the message dictionary can contain one or more variable fields that are replaced with application-specific values at run time. In a DISPLAY command, the PARMS parameter can be used to code replacement parameters for each variable field in a specified message.

Within the message definition in the dictionary, symbolic parameters are identified by an ampersand (&) followed by a 2-digit numeric identifier. These identifiers can appear in any order. The position of the replacement values in the PARMS parameter must correspond directly to the 2-digit numeric identifiers in the message; the first value corresponds to &01, the second to &02, and so forth. For example, assume that the stored message text is as follows:

```
THIS IS TEXT &01 AND &03 OR &02
```

The PARMs parameter reads PARMs=('A','B','C'). The resulting text would read as follows:

```
THIS IS TEXT A AND C OR B
```

If the message is defined in the dictionary with more than one text line, only the first line appears in the map's message field.

If the message is defined in the dictionary with a destination of TERMINAL, the message will be redisplayed at the user's terminal when control exits from the CA-ADS application.

Examples: The examples below are based on the sample applications shown in 15.2.1, “Application thread” earlier in this chapter where dialog UPDATEO updates or erases all ORDOR records associated with a CUSTOMER record that is retrieved by dialog UPDATEC.

Example 1: Retrieving records

The following sample premap process from UPDATEO retrieves the ORDOR records to be changed. The DISPLAY command is used to display the dialog's map with a message informing the user of the processing status:

```
READY.  
OBTAIN NEXT ORDOR WITHIN CUSTOMER-ORDER.  
IF DB-END-OF-SET  
THEN  
    DISPLAY MESSAGE TEXT IS  
    'CUSTOMER HAS NO ORDERS. HIT 'CLEAR' TO EXIT.'  
ELSE  
    DISPLAY MESSAGE CODE IS 900101  
    PARMs = (ORD-NUMBER,'ORDERS').
```

Example 2: Erasing records

The following sample response process from UPDATEO erases a retrieved ORDOR record. DISPLAY CONTINUE is used to return control to the dialog's premap process, which retrieves the next ORDOR record:

```
READY USAGE-MODE IS UPDATE.  
ERASE ORDOR ALL MEMBERS.  
DISPLAY CONTINUE.
```

15.5 EXECUTE NEXT FUNCTION

Purpose: Passes control in a dialog that is associated with an application function.

Syntax

►—— EXECute next function —— . —————►◄

Usage: EXECUTE NEXT FUNCTION is appropriate for use in applications defined by using the CA-ADS application compiler (ADSA).

When the user selects a response that is valid for a dialog function at runtime, the function associated with the response is established as the next function to be executed. The EXECUTE NEXT FUNCTION command initiates execution of that function. Control is passed to the function by means of the control command associated with the application response during application compilation.

Considerations

- An EXECUTE NEXT FUNCTION command in a dialog that is not associated with an application function is processed by the CA-ADS runtime system as a DISPLAY command. The runtime system displays the following message in the map's message field:
DC177018 PLEASE SELECT NEXT FUNCTION
 - The EXECUTE NEXT FUNCTION command executes the function that is invoked by the application response specified in the AGR-CURRENT-RESPONSE field of the ADSO-APPLICATION- GLOBAL-RECORD. Note that the response is moved into AGR-CURRENT-RESPONSE when the user selects an application response.
 - Premap and response process commands can modify the value of AGR-CURRENT-RESPONSE, thereby modifying the function executed by the EXECUTE NEXT FUNCTION command.
 - For more information on the AGR-CURRENT-RESPONSE field, see "ADSO-APPLICATION-GLOBAL-RECORD" in Appendix A, "System Records."
 - The premap process of a mapless dialog must move a valid application response to AGR-CURRENT-RESPONSE before issuing an EXECUTE NEXT FUNCTION command.
 - If AGR-CURRENT-RESPONSE is modified by a process command, the runtime system does not perform security checking.
- The effect of the EXECUTE NEXT FUNCTION command is shown in "Runtime flow of control" in Chapter 4, "CA-ADS Runtime System."

Example: In this example, control passes to the next function in the CA-ADS application after the end-of-set condition is reached:

```
WHILE NOT DB-END-OF-SET
  REPEAT.
    OBTAIN NEXT ORDOR WITHIN CUST-ORDOR.
    .
    .
    .
  END.
EXECUTE NEXT FUNCTION.
```

Because EXECUTE NEXT FUNCTION is used to pass control in this example, the CA-ADS runtime system determines which function to execute next.

15.6 INVOKE

Purpose: Passes control to a specified dialog in the current application and implicitly establishes the next lower level in the application thread.

Syntax

```

►► INvoke [ NOSAVE ] dialog-name . ►◄

```

Parameters

NOSAVE

Specifies that database currencies are not saved for the dialog issuing the INVOKE command.

dialog-name

Specifies either the name of a variable data field containing the dialog name to which control passes or the dialog name itself, enclosed in single quotation marks.

Usage

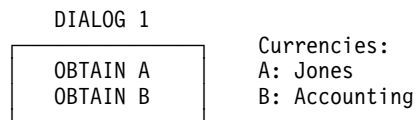
Considerations

- The load module for the named dialog must be available at run time.
- The dialog that issues the INVOKE command remains operative.
- A lower level dialog can return control to the dialog by issuing a RETURN command.
- The issuing dialog's database currencies are saved and available to the dialog receiving control, unless the NOSAVE option is specified.

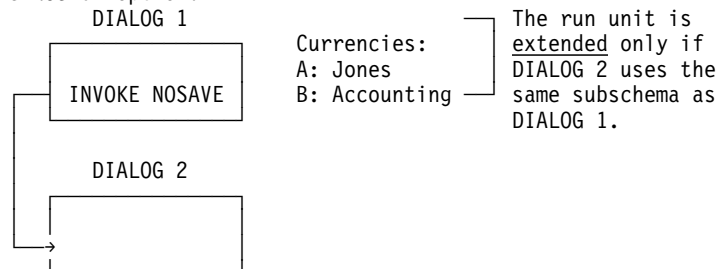
When a dialog that issued an INVOKE NOSAVE command regains control from a lower level dialog or program, database currencies are dependent upon whether or not the run unit was extended. The following diagram shows how currencies are affected when the NOSAVE option is used in extended and nonextended run units.

Currency settings of extended and nonextended run units

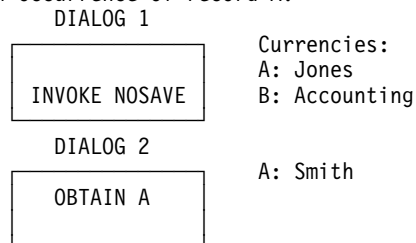
1. DIALOG 1, which uses subschema SS1, obtains values for records A and B:



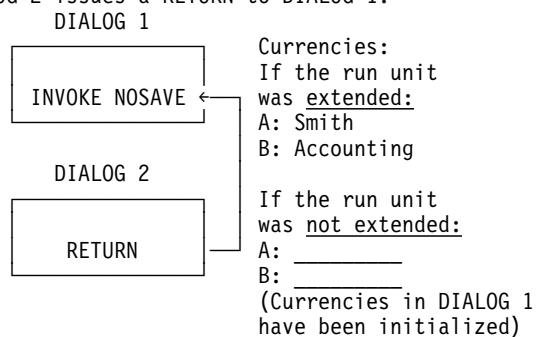
2. DIALOG 1 invokes DIALOG 2 using the NOSAVE option:



3. DIALOG 2, which can use DIALOG 1's record buffers and currencies, obtains a new occurrence of record A:



4. DIALOG 2 issues a RETURN to DIALOG 1:



Considerations for using NOSAVE

- When the dialog that issues the command regains control from a lower level dialog and the run unit is extended by the INVOKE command, currencies are set to those of the most recent dialog returning control.
- When the dialog that issues the command regains control from a lower level dialog and the run unit is not extended by the INVOKE command, currencies are set to the original currencies available to the dialog when it became operative in the application thread.

►► Information about extended run units can be found in Chapter 4, “CA-ADS Runtime System.”

Example: In the sample applications shown in 15.2.1, “Application thread” earlier in this chapter, dialog RDCUST# prompts the user for the CALC key of a CUSTOMER record to be retrieved. RDCUST# passes control to dialog UPDATEC, which retrieves and displays the record, and then modifies or erases it as instructed by the user. RDCUST# uses the following response process to pass control to UPDATEC:

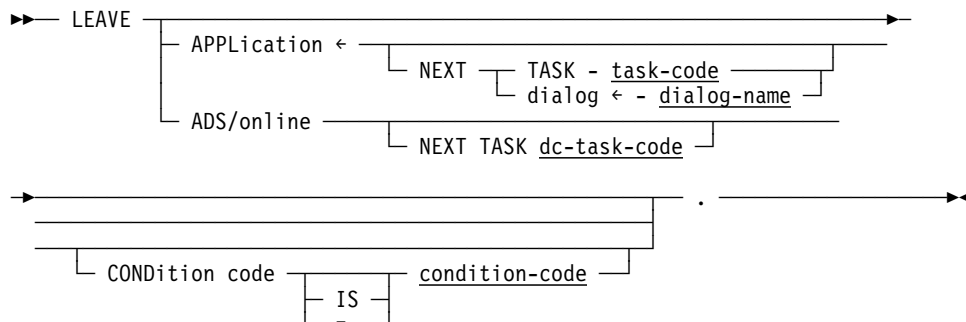
```
INVOKE 'UPDATEC'.
```

Because RDCUST# uses the INVOKE command to pass control, processing can return to the RDCUST# mapout operation following completion of UPDATEC processing. This allows the user to update multiple CUSTOMER records in one CA-ADS runtime session.

15.7 LEAVE

Purpose: Terminates the current application thread or terminates the current CA-ADS runtime session.

Syntax



Parameters

APPLICATION

Terminates the current application and passes control as specified by NEXT TASK or NEXT dialog.

LEAVE is the equivalent of LEAVE APPLICATION.

NEXT TASK task-code

Passes control to an application as defined on the Task Codes screen of the application compiler.

Task-code specifies an application task code, as defined on the **Task Codes** screen of the application compiler. *Task-code* is either the name of a variable field containing the task code or the task code itself, enclosed in single quotation marks.

NEXT dialog dialog-name

Specifies the name of a mainline dialog to which control passes. If the keyword TASK or dialog is not specified, dialog is the default.

Dialog-name is either the name of a variable data field that contains the dialog name or the dialog name itself, enclosed in single quotation marks.

The load module for the named dialog must be stored in the data dictionary.

ADS/online

Terminates the current application and the current CA-ADS session. Control returns to CA-IDMS/DC or CA-IDMS/UCF (DC/UCF).

NEXT TASK dc-task-code

(Online only) Passes control to another DC/UCF task.

Dc-task-code is either the name of a variable field containing the DC/UCF task or the task name itself, enclosed in single quotation marks.

Dc-task-code must be defined with the NOINPUT parameter, which specifies that only a task code, and no additional data, is expected.

CONDition code IS condition-code

(Batch OS/390 only) Clause introducing a completion code for the current job step.

The completion code can be tested using the COND parameter in the job control language (JCL).

Condition-code is either the name of a variable field containing the condition code or the number itself, expressed as a numeric literal.

IS or = are optional keywords and have no effect on processing.

Usage

Effects of issuing LEAVE

- All operative dialogs in the terminating application become nonoperative.
- All database currencies are released.
- All record buffers are freed.

Example: Dialog UPDATEC, shown in Example 1 in the earlier diagram, includes the following response process, which allows the terminal operator to terminate the application thread:

LEAVE APPLICATION.

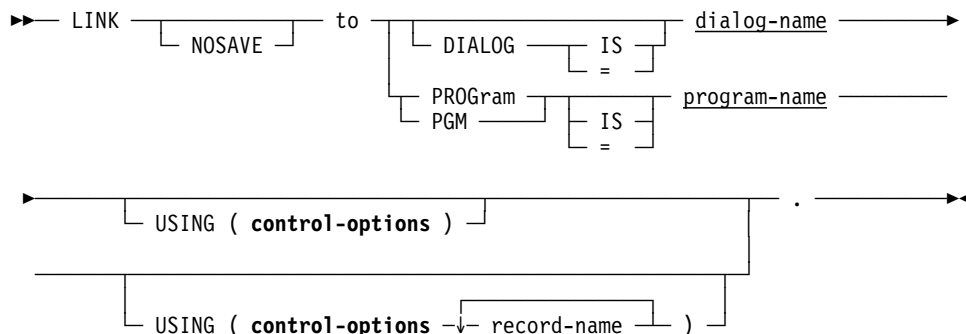
When the above response process executes, control passes to the Dialog Selection screen. The user can then select the next mainline dialog to be executed.

15.8 LINK

Purpose: Specifies the next dialog executed in a current application.

LINK is also used to request execution of a COBOL, PL/I, or Assembler program.

Syntax:



Expansion of control-options



Parameters

NOSAVE

Specifies that the database currencies for the dialog that issues the LINK command are not saved.

When a dialog that issues a LINK NOSAVE command regains control from a lower level dialog or program, its database currencies are set as follows:

- If the LINK command extends a run unit, the dialog's currencies are passed back up to the dialog or program to which the linking dialog passed control.
- If the LINK command does not extend a run unit, the dialog's database currencies are reinitialized to whatever they were when the dialog gained control.

DIALOG IS dialog-name

Specifies the name of the dialog to which control passes.

Dialog-name is either the name of a variable data field that contains the dialog name or the dialog name itself, enclosed in single quotation marks.

The load module for the named dialog must be stored in the data dictionary of load library.

IS or = are optional keywords and have no effect on processing.

PROGram IS program-name

Specifies the name of the COBOL, PL/I, or Assembler program to which control is passed.

Program-name is either the name of a variable data field that contains the program name or the program name itself, enclosed in single quotation marks.

The load module for the named program must be defined under DC/UCF as a program. The program can be defined in any of the following ways:

- At system generation by means of the PROGRAM statement
- In the IDD by means of the DDDL PROGRAM statement
- Under DC/UCF by means of the DCMT VARY DYNAMIC PROGRAM master terminal command

PGM can be used in place of PROGRAM.

USING control-options

Identifies the control options to be used.

Expanded syntax for control-options is shown above immediately following the LINK syntax.

Multiple parameters in the USING clause must be separated by blanks or commas. The record and control block names must be specified in the same order in which they are defined in the user program.

The SQLSSI parameter is used when passing a global cursor from a CA-ADS dialog to a user program. SQLSSI is a record that contains the SQL session identifier which is assigned when the dialog's transaction started. This record is copied into the dialog automatically, so the user does not need to add it to the dialog. The user program must have a record in its "linkage" section defined with the SQLSESS datatype.

record-name

Specifies the data that is passed to the named user program.

MAP-CONTROL

Passes the map request block of the original CA-ADS dialog to the lower level dialog.

The lower level dialog must specify the same map as the calling dialog. The version number and date/time stamp for both maps must be identical. If the maps differ, the application abends.

The keyword MAP_CONTROL may be used in place of MAP-CONTROL.

SUBSCHEMA-CONTROL

Extends a calling dialog's run unit to a lower level dialog. The runtime system ignores any differences between the two dialog's subschemas, schemas, and area ready modes. On return to the calling dialog, the run unit is unconditionally extended upward.

The dialog to which control is extended is not allowed to access a record or set not defined in the original dialog's subschema. Such an attempt causes an abend at runtime.

The keyword SUBSCHEMA_CONTROL may be used in place of SUBSCHEMA-CONTROL.

Usage

Control passed to a specified dialog: When control is passed to a specified dialog by means of a LINK command, the next lower level in the application thread is implicitly established and a nested structure is created.

Considerations

- The dialog issuing the LINK command becomes the top of the nested structure and remains operative.

If an application response passes control by means of a LINK command, the function from which the response was selected becomes the top of a nested structure.

- A LINK command within a nested structure establishes the top of a lower nested level.
- Dialogs within a nested structure can issue any of the control commands.

A RETURN command cannot pass control higher than the top of the lowest nested level that is operative in the application thread.

- The dialog issuing a LINK command expects control to return to the command following the LINK instruction.
- The issuing dialog's database currencies are saved and are available to the dialog when it regains control, unless the NOSAVE option is used.

When the dialog that issued a LINK NOSAVE command regains control from a lower level dialog or program, the database currencies set depend on whether or not the run unit was extended.

Refer to the LINK command syntax rules that follow this discussion for currency settings of extended and nonextended run units.

►► Information about extended run units can be found in Chapter 4, "CA-ADS Runtime System."

Control passed to a user program: When a LINK command specifies a user program, control passes outside the CA-ADS environment and temporarily suspends CA-ADS sessions.

Considerations

- The LINK command must explicitly specify any data to be passed to the user program, including the subschema control block, the map request block, and any records used in the program's processing.
- A user program has the option of using the calling dialog's run unit.

If the LINK command does not contain subschema-control in its USING list, the user program cannot access its calling dialog's run unit. The user program can access a database by binding a run unit and establishing its own currencies. This run unit will be bound concurrently with the dialog's run unit.

If the LINK command contains subschema-control in its USING list, the dialog's run unit is passed to the user program. Any database records to be shared with the dialog should be passed in the USING RECORD list.

- A user program must return control to CA-ADS by means of a DC RETURN statement.

When the user program issues the DC RETURN statement, the suspended CA-ADS session resumes and control passes to the command following the LINK command.

The format of the DC RETURN statement varies based on whether the program has previously issued a DC RETURN statement that specified a next task code other than ADSR, as follows:

- If the program has previously issued a DC RETURN statement that specified a next task code other than ADSR, the DC RETURN statement that returns control to CA-ADS must have the following format:

```
DC RETURN NEXT TASK CODE ADSR.
```

DC RETURN NEXT TASK CODE will end the task and rollback any open run unit, whether it was bound by the user program or passed from the calling dialog.

ADSR is the default task code that invokes ADSOMAIN with no input. The task code can be changed by means of the DC/UCF system generation TASK statement.

►► For more information on specifying the task code for the CA-ADS runtime system, refer to *CA-IDMS System Generation*.

- If the program has not previously issued a DC RETURN statement that specified a next task code other than ADSR, the DC RETURN statement that returns control to CA-ADS can have the following format:

```
DC RETURN
```

- A dialog with a standard subschema can link to a dialog with an LRF subschema using subschema control. However, if the lower-level dialog makes an LR call, a status of 0063 is returned; in this case, the status is equivalent to a status of 2008.

To use the LINK command effectively in conjunction with user programs, refer to the online programming techniques presented in the *CA-IDMS DML Reference* for the appropriate language.

Examples: Example 1: Passing control to a lower level dialog

Dialog UPDATEC, shown in Example 1 of 15.2.1, “Application thread” earlier in this chapter, uses the response process listed below to pass control to dialog UPDATEO.

UPDATEO obtains an ORDOR record for the current CUSTOMER, requests modifications, and updates the record in the database. When UPDATEO returns control to dialog UPDATEC, processing resumes with the DISPLAY command that follows the LINK command:

```
LINK TO DIALOG 'UPDATEO'.  
DISPLAY MESSAGE TEXT IS  
'CUSTOMER ORDER HAS BEEN CHANGED'.
```

Example 2: Passing control to a COBOL program

The following statement from the premap process associated with dialog UPDATEC passes control to the COBOL program LOOKUP. LOOKUP uses the subschema control block and CUSTOMER record buffer from UPDATEC to check the status of the current customer:

```
LINK PROGRAM 'LOOKUP'  
USING (SUBSCHEMA-CONTROL,CUSTOMER).
```

Example 3: Extending the current map session

In this example, ERRCHK is a dialog that contains special error-checking and validating routines. ERRCHK uses the same map as the calling dialog. The LINK command passes current map attributes and data to ERRCHK.

When ERRCHK finds errors, it:

- Sets the appropriate fields in error by modifying error attributes for the map.
- Returns control to the calling dialog. The error attributes are returned along with current map data.

The sample LINK statement that passes control to ERRCHK is:

```
LINK TO DIALOG 'ERRCHK'  
USING (MAP-CONTROL).
```

Example 4: Extending the current run unit

In this example:

- The calling dialog uses subschema EMPSS01. This subschema contains records EMPLOYEE and DEPARTMENT.

- The LINK command unconditionally extends the current run unit to dialog UPDATE, which is a mapless dialog containing update logic for records EMPLOYEE and DEPARTMENT.
- The USING statement bypasses the checking of the subschema and ready modes when passing the run unit.
- Dialog UPDATE updates the database and then returns control to the calling dialog.

The sample LINK statement that passes control to UPDATE is:

```
LINK TO DIALOG 'UPDATE'  
USING (SUBSCHEMA-CONTROL).
```

Example 5: COBOL program that was passed the dialog's subschema-control

The following example shows a LINKed-to COBOL program that was passed the dialog's SUBSCHEMA-CONTROL.

```
ENVIRONMENT DIVISION.  
    PROTOCOL. IDMS-RECORDS MANUAL.  
WORKING-STORAGE SECTION.  
    01 COPY IDMS SUBSCHEMA-NAMES.  
    01 COPY IDMS RECORD <the name of each database record that is  
                           needed but was not passed in the USING list>  
LINKAGE SECTION.  
    01 COPY IDMS SUBSCHEMA-CTRL.  
    01 COPY IDMS RECORD <the name of each database record that is  
                           passed in the LINK command>  
PROCEDURE DIVISION.  
    BIND <the name of each database record that is needed but was  
         not passed in the LINK command>
```

15.9 READ TRANSACTION

Purpose: (CA-ADS/Batch only) Terminates the current process, performs a mapin operation, and then selects the next application function or response process to be executed.

Syntax

►►—— READ TRANsaction ———┐ OUTput ┘ . —————►►

Parameters

OUTput

Specifies that the file is to be opened as an input/output file.

Usage

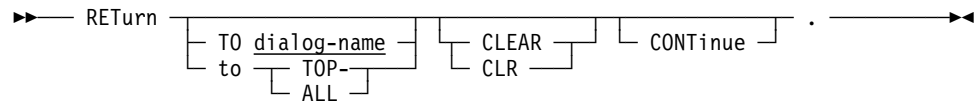
Considerations

- OUTPUT must be specified in the first READ TRANSACTION command for a VSAM entry-sequenced data set (ESDS) that is to be opened for both input and output.
- OUTPUT is ignored if the file is already opened; if the file is not a VSAM ESDS file, the application abends.
- If the current record's response field selects an immediately executable function that is not the same as the current function, the runtime system passes control to the newly selected function. The next time a mapin operation is performed for the file, the runtime system immediately maps in the record.
- On a mapin operation, the runtime system automatically opens the file being read if the file is not already opened.

15.10 RETURN

Purpose: Passes control to a higher level dialog or function in the application thread.

Syntax



Parameters

TO dialog-name

Introduces the name of a higher level dialog to which control passes.

Dialog-name can be the name of a variable data field that contains the dialog name or the dialog name itself, enclosed in single quotation marks.

to TOP

Specifies the highest level to which control can pass.

ALL can be used in place of to TOP.

CLEAR

Specifies that record buffers are reinitialized and currencies are released for the dialog receiving control.

CLEAR is ignored if the receiving dialog is at the top of a nested application structure.

CLR can be used in place of CLEAR.

CONTinue

Specifies that control returns to the first command in the premap process of the dialog receiving control.

If CONTINUE is not specified, control returns to the mapout operation of the dialog that receives control. If the receiving dialog is at the top of a nested application structure, CONTINUE is ignored.

Usage

Considerations

- The dialog or function receiving control must be operative.
- The dialog that issues the RETURN command becomes nonoperative as do any operative dialogs or functions on a level between the issuing and receiving dialogs or functions.
- All database currencies established by the dialog issuing the RETURN command are released.

- A RETURN command cannot pass control higher than the top of the lowest level nested application structure created by a LINK command.
- The named dialog must not be higher than the top of the nested application structure in which the issuing dialog participates.
- If the named dialog is operative at more than one higher level, control passes to the lowest level operative dialog with the specified name.
- If the issuing dialog participates in a nested application structure, control returns to the top of the nested structure.
- If the issuing dialog does not participate in a nested structure, control returns to the mainline dialog at the top of the application thread.
- If a RETURN statement does not specify a receiving dialog, control passes to the next higher level dialog or function.
- If the mainline dialog at the top of an application thread issues a RETURN command, the RETURN command is treated as a LEAVE APPLICATION command.
- A dialog that receives control at the top of a nested structure resumes execution at the command that follows the LINK command.
- RETURN can pass control within a nested application structure to any operative dialog that passed control with an INVOKE command.
- If the issuing dialog is not in a nested structure, RETURN can pass control to any higher level operative dialog or function, or directly to the top of the application structure.
- The application developer can specify whether the dialog receiving control resumes execution with its premap process or with its mapout operation.
- The application developer can also request reinitialized record buffers for the dialog that receives control.

Examples: The examples below show the use of the RETURN command in response processes from dialogs used in the two sample applications shown in 15.2.1, “Application thread” earlier in this chapter:

Example 1: Using RETURN with the LINK command

In Example 1 of 15.2.1, “Application thread” earlier in this chapter, dialog UPDATEEC passes control to the dialog UPDATEO by means of a LINK command. The following response process from dialog UPDATEO returns control to the command following the LINK command in dialog UPDATEEC:

```
READY USAGE-MODE IS UPDATE.  
MODIFY ORDOR.  
RETURN.
```

Example 2: Using RETURN with the INVOKE command

In Example 2 of 15.2.1, “Application thread” earlier in this chapter, dialog UPDATEC passes control to the dialog UPDATEO by means of an INVOKE command. The following response process from dialog UPDATEO returns control to the mainline dialog RDCUST# and reinitializes the record buffers associated with RDCUST#:

```
READY USAGE-MODE IS UPDATE.  
MODIFY ORDOR.  
RETURN TOP CLEAR.
```

Example 3: Transferring control within the same level

In Example 2 of 15.2.1, “Application thread” earlier in this chapter, dialog UPDATEO provides the ability to transfer to dialog ADDORDR. ADDORDR prompts the user for new order information. The following response process from dialog ADDORDR adds a new ORDOR record to the database and returns control to the mapout operation of dialog UPDATEC:

```
READY USAGE-MODE IS UPDATE.  
STORE ORDOR.  
RETURN.
```

15.11 TRANSFER

Purpose: Passes control to a specified dialog at the same level in the application structure.

Syntax:

► TRANSfer NOFinish to dialog-name .

Parameters

NOFinish

Specifies that the current run unit is to be extended.

► For information about extended run units, see Chapter 4, “CA-ADS Runtime System.”

to dialog-name

Either the name of a variable data field that contains the dialog name to which control passes or the dialog name itself, enclosed in single quotation marks.

The load module for the named dialog must be stored in the data dictionary.

Usage

Considerations

- When specified along with the NOFINISH option, TRANSFER can extend the current run unit.
- When TRANSFER is specified without NOFINISH, a dialog that issues a TRANSFER command becomes nonoperative.
- The receiving dialog or function replaces the issuing dialog in the application thread.
- The receiving dialog or function has access to database currencies established by dialogs at higher levels in the application thread and to the contents of global records and of any records whose buffers were established by dialogs at higher levels in the application thread.
- A dialog can transfer control to itself.
- The copy of the dialog receiving control acquires newly initialized record buffers.
- When a dialog transfers control to itself, the FIRST-TIME status is reset.
 - FIRST-TIME status is discussed in Chapter 8, “Conditional Expressions.”

Examples: The following examples use the TRANSFER statement to pass control to a dialog at the same level.

Example 1: Using the dialog name

In Example 2 of 15.2.1, “Application thread” earlier in this chapter, dialog UPDATEO passes control to dialog ADDORDR by means of the following statement:

```
TRANSFER TO 'ADDORDR'.
```

Example 2: Using a variable data field to transfer control

In this example, control passes either to dialog ADDORDR or to dialog ORDCOUNT, depending on the outcome of the OBTAIN command:

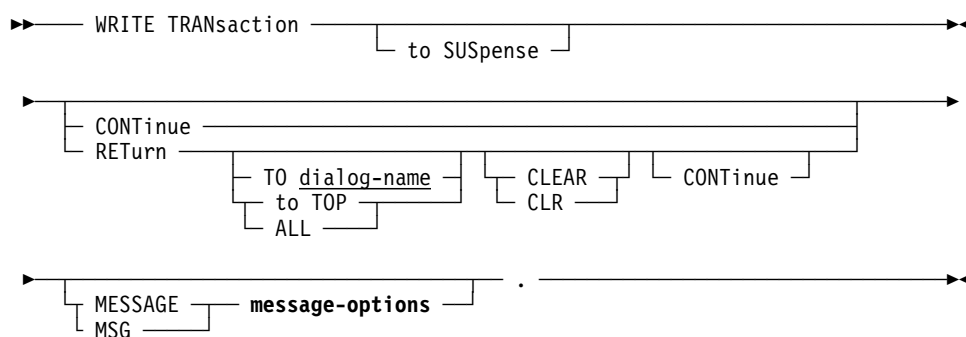
```
OBTAIN NEXT ORDOR WITHIN CUST-ORDER.  
IF DB-END-OF-SET  
THEN  
    MOVE 'ADDORDR' TO NEXT-DIALOG.  
ELSE  
    MOVE 'ORDCOUNT' TO NEXT-DIALOG.  
TRANSFER TO NEXT-DIALOG.
```

15.12 WRITE TRANSACTION

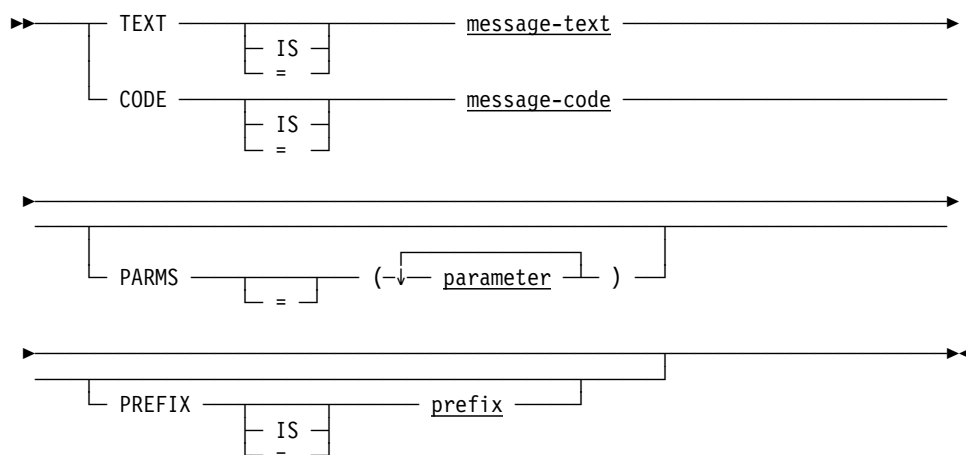
Purpose: (CA-ADS/Batch only) Performs the following sequence of functions:

1. Terminates the current process
2. Performs a mapout operation
 - If the dialog's current input file record contains no errors and the keyword SUSPENSE is not included in the command, the mapout writes a record to the dialog's associated output file, according to the output file map definition.
 - If the input file record contains errors or the keyword SUSPENSE is included in the command, the mapout writes the input record to the dialog's suspense file and sends applicable error messages to the log file.
3. Passes control within the application. Control can be passed to:
 - The dialog's premap process or mapin operation
 - A higher level dialog or application function

Syntax



Expansion of message-options



Parameters

to SUSPense

Specifies that the dialog's input record is written to the suspense file even if it does not contain errors. Nothing is written to the dialog's output file.

CONTInue

Specifies that control is passed to the dialog's premap process after mapout operation.

RETurn

Specifies that control is returned to a higher level dialog or application function after mapout operation.

TO dialog-name

Either the name of a variable data field that contains the dialog name to which control is passed or the dialog name itself, enclosed in single quotation marks.

to TOP

Specifies the highest level to which control can pass.

ALL can be used in place of to TOP.

CLEAR

Specifies that record buffers are reinitialized and currencies are released for the dialog receiving control.

CLEAR is ignored if the receiving dialog is at the top of a nested application structure.

CLR can be used in place of CLEAR.

CONTInue

Specifies that control returns to the first command in the premap process of the dialog receiving control. If not specified, control returns to the mapout operation of the dialog that receives control. If the receiving dialog is at the top of a nested application structure, CONTINUE is ignored.

If neither CONTINUE nor RETURN is specified, control passes to the dialog's mapin operation. The runtime system maps the next record into variable storage, then selects the next application function or dialog response process to be executed.

Note: For applications defined using the application compiler, the runtime system first examines the current record's response field. If the field selects an immediately executable function that is not the same as the current function, the runtime system passes control to the selected function. The next time a mapin operation is performed for the file, the runtime system immediately maps in the record.

MESSAge message-options

Identifies message to be displayed.

MSG can be used in place of MESSAGE.

Expanded syntax for message-options is shown above immediately following the CONTINUE syntax.

TEXT IS message-text

Specifies the text of a message to be displayed in an online map's message field or sent to a batch application and a system log file.

Message-text specifies either the name of a variable data field containing the message text or the text string itself, enclosed in single quotation marks.

The text string can contain up to 240 displayable characters.

IS or = are optional keywords and have no effect on processing.

CODE IS message-code

Specifies the message dictionary code of a message to be displayed in an online map's message field or sent to the log file in a batch application.

In a batch application, the message is also sent to the operator, if directed by the destination specified in the dictionary.

Message-code specifies either the name of a variable data field that contains the message code or the 6-digit code itself, expressed as a numeric literal.

IS or = are optional keywords and have no effect on processing.

PARMS = parameter

Specifies a replacement parameter for each variable field in the stored message identified by *message-code*.

Up to nine replacement parameters can be specified for a message. Multiple parameters must be specified in the order in which they are numbered and separated by blanks or commas.

Parameter specifies either the name of an EBCDIC or unsigned zoned decimal variable data field that contains the parameter value or the actual parameter value, enclosed in single quotation marks.

The parameter value must contain displayable characters. At run time, each variable data field in a stored message expands or contracts to accommodate the size of its replacement parameter. A replacement parameter can be a maximum of 240 bytes.

PREFIX IS prefix

Overrides the default prefix of a dialog and a map.

Prefix must either specify an EBCDIC or unsigned zoned decimal variable data field that contains a 2-character prefix or the 2-character prefix itself, enclosed in single quotation marks

IS or = are optional keywords and have no effect on processing.

Usage**Considerations**

- The named dialog must not be higher than the top of a nested application structure in which the issuing dialog participates.

- If the named dialog is operative at more than one higher level, control passes to the lowest level dialog with the specified name.
- If the write operation results in a physical output-error condition, the application terminates.
- A WRITE TRANSACTION command can be issued in a dialog that is not associated with an output file. In this case, the command is used only to write an input record to the suspense file. If the input record is not in error, nothing is written to the suspense file.
- The WRITE TRANSACTION command also allows specification of a message to be sent to the log file or to the operator's console.
- The destination of the message depends on the routing codes specified using ADSOBSYS or at run time in a control statement.
- If the issuing dialog participates in a nested application structure, control returns to the top of the nested structure.
- If the issuing dialog does not participate in a nested structure, control returns to the mainline dialog at the top of the application thread.
- If a RETURN statement does not specify a receiving dialog or TOP, control passes to the next higher level dialog or function.
- If the mainline dialog at the top of an application thread issues a RETURN command, the RETURN command is treated as a LEAVE APPLICATION command.
- Up to nine replacement parameters can be specified for a message.
- Multiple parameters must be separated by blanks or commas.
- Multiple parameters must be specified in the order in which they occur in the stored message.

Chapter 16. Database Access Commands

16.1 Overview	16-3
16.2 Navigational DML	16-5
16.2.1 Overview of navigational database access	16-5
16.2.2 Use of native VSAM data sets	16-7
16.2.3 Record locking	16-9
16.2.4 Suppression of record retrieval locks	16-10
16.2.5 Overview of ACCEPT	16-12
16.2.6 ACCEPT DB-KEY FROM CURRENCY	16-12
16.2.7 ACCEPT DB-KEY RELATIVE TO CURRENCY	16-14
16.2.8 ACCEPT PAGE-INFO	16-16
16.2.9 ACCEPT STATISTICS	16-17
16.2.10 BIND PROCEDURE	16-19
16.2.11 COMMIT	16-20
16.2.12 CONNECT	16-22
16.2.13 DISCONNECT	16-25
16.2.14 ERASE	16-27
16.2.15 Overview of FIND/OBTAIN	16-30
16.2.16 FIND/OBTAIN CALC	16-31
16.2.17 FIND/OBTAIN CURRENT	16-33
16.2.18 FIND/OBTAIN DB-KEY	16-34
16.2.19 FIND/OBTAIN OWNER	16-37
16.2.20 FIND/OBTAIN WITHIN SET/AREA	16-38
16.2.21 FIND/OBTAIN WITHIN SET USING SORT KEY	16-42
16.2.22 GET	16-44
16.2.23 KEEP	16-46
16.2.24 KEEP LONGTERM	16-47
16.2.25 MODIFY	16-53
16.2.26 READY	16-55
16.2.27 RETURN DB-KEY	16-57
16.2.28 ROLLBACK	16-59
16.2.29 STORE	16-60
16.3 Logical Record Facility commands	16-64
16.3.1 Overview of LRF database access	16-64
16.3.2 WHERE clause	16-65
16.3.3 Conditional expression	16-65
16.3.4 Comparison expression	16-66
16.3.5 ERASE	16-68
16.3.6 MODIFY	16-69
16.3.7 OBTAIN	16-70
16.3.8 ON command	16-71
16.3.9 STORE	16-75

16.1 Overview

A CA-ADS application can access the CA-IDMS/DB database by using navigational DML or SQL DML.

Navigational DML: CA-ADS navigational DML is used to retrieve and update database or VSAM records and perform database control functions. Navigation DML commands can be used in process logic to store, retrieve, modify, and delete data in a non-SQL defined database, using a standard subschema or a Logical Record Facility (LRF) subschema.

When using LRF, the application developer selects a predefined path that meets the dialog's data requirements and codes simple database requests in dialog process logic. Database navigation is defined in the path, not in the process.

SQL DML: In a CA-ADS application, SQL DML can be used to retrieve and update data defined with:

- Records in non-SQL defined databases (associated with an SQL schema)
- Tables in SQL-defined databases

►► For more information about using SQL DML statements, see *SQL Self-Training Guide* and *CA-IDMS SQL Programming*.

Navigational DML and LRF commands used in the CA-ADS environment are summarized in the following two tables. Documentation of command syntax appears later in this chapter.

Summary of navigational DML commands

Command	Purpose
ACCEPT	Moves database keys page info statistics from the database management system to a dialog
BIND PROCEDURE	Establishes communication from a dialog to a DBA-written procedure
COMMIT	Writes checkpoints to the journal file and releases locks held on database records
CONNECT	Connects member records to sets
DISCONNECT	Disconnects member records from sets
ERASE	Erases records from the database
FIND	Locates records in the database

Command	Purpose
GET	Copies record contents from the database to a dialog's record buffers
KEEP	Places locks on records
MODIFY	Replaces records in the database with the contents of a dialog's record buffers
OBTAIN	Locates records in the database and copies their contents to a dialog's record buffers
READY	Prepares database areas for processing
RETURN DB-KEY	Retrieves index entries without the associated record (used only with the Sequential Processing Facility and with system-owned indexed records)
ROLLBACK	Requests recovery of the database
STORE	Adds a record to the database

Summary of LRF commands

Command	Purpose
ERASE	Deletes Logical Record Facility record occurrences
MODIFY	Changes field values in Logical Record Facility record occurrences
OBTAIN	Retrieves Logical Record Facility record occurrences
ON	Performs additional processing based on the outcome of conditional testing of Logical Record Facility record access
STORE	Stores a new occurrence of a Logical Record Facility record

16.2 Navigational DML

Each navigational DML command is presented alphabetically after the overview of navigational database access.

16.2.1 Overview of navigational database access

To use navigational DML commands effectively in a process, the application developer should be familiar with database programming concepts. These concepts are discussed in detail in the *CA-IDMS Navigational DML Programming*.

Considerations: The following special considerations apply to accessing the database in the CA-ADS environment:

- Before coding database commands, the application developer must be familiar with the characteristics of the subschema associated with the dialog. The subschema specifies the elements, records, sets, and areas available to the dialog. The subschema also includes the default usage modes for the database areas and specifies any restrictions on the use of database commands.
- Each database command can be coded any number of times within a process.
- If a READY command specifies the same area more than once within a process, the usage mode specified in the last READY statement applies to the specified area for the entire process.

The READY command is executed when the first DML command is encountered. If an invalid (non-zero) error status is returned from the READY or BIND processing, the dialog aborts. Process code cannot intercept these errors.

- At runtime, CA-ADS automatically initializes a buffer for each record type associated with the mainline dialog. Subsequent dialogs that access the same record type use the existing record buffer unless a reinitialized buffer for the record is requested by using the Records and Tables screen during dialog compilation.
 - ▶▶ The Records and Tables screen is described in "Records and Tables screen" in Chapter 3, "CA-ADS Dialog Compiler (ADSC)."
- To enable proper positioning and movement through the database during the execution of an application, the CA-ADS runtime system automatically maintains database keys for the records that are accessed by a dialog as shown in the following table.

Record	Description
Current of run unit	The most recently accessed record occurrence
Current of record type	The most recently accessed occurrence of each record type

Record	Description
Current of set type	The most recently accessed record occurrence (owner or member) of each set
Current of area	The most recently accessed record occurrence in each area

- Database commands use and update currencies, as listed in the currency chart below.

CA-ADS saves or releases the currencies established during dialog execution based on the command used to pass control to the next function or dialog.

►► For a discussion of currency action, see Chapter 15, “Control Commands.”

Database command	Currency updated by successful execution				Successful execution
	Run unit	Record	Set	Area	
ACCEPT*	X	X	X	X	None
IF*	X		X		None
FIND/OBTAIN DB-KEY					All
FIND/OBTAIN CURRENT*	X	X	X	X	All
FIND/OBTAIN WITHIN SET			X		All
FIND/OBTAIN WITHIN AREA				X ²	All
FIND/OBTAIN OWNER			X		All
FIND/OBTAIN CALC					All
FIND/OBTAIN DUPLICATE		X			All
FIND/OBTAIN USING SORT KEY			X		All
GET	X				None
STORE			X ³		All
MODIFY	X				None ⁴

Database command	Currency updated by successful execution				Successful execution
	Run unit	Record	Set	Area	
ERASE	X				Nullifies of all record types and sets involved
CONNECT		X	X		Run unit, set
DISCONNECT		X			Nullifies currency of object set; updates current of run unit and area
KEEP*	X	X	X	X	None
COMMIT					None
COMMIT ALL					Nullifies all currencies
ROLLBACK					Nullifies all currencies
ROLLBACK CONTINUE					Nullifies all currencies
FINISH					Nullifies all currencies

Notes:

* Uses only one currency as determined by command format.

² Required for NEXT and PRIOR formats only.

³ All in which record type participates as an automatic member.

⁴ Except in the case of a sorted set.

16.2.2 Use of native VSAM data sets

Native VSAM data sets can be defined in a CA-IDMS/DB database schema and accessed by CA-ADS database commands as if they were standard database files. CA-IDMS/DB supports all three types of VSAM data sets: key sequenced (KSDS), entry sequenced (ESDS), and relative record (RRDS).

Existing VSAM data structures are accessed by equating them to CA-IDMS/DB structures in the schema. The dialog issues standard process command statements for the equivalent CA-IDMS/DB structures and the DBMS converts these statements to native VSAM access requests for the appropriate VSAM structures.

When a dialog's subschema includes records in a native VSAM file, process code for the dialog is affected in the following ways:

- The set status condition cannot be used with sets defined for native VSAM data records.
 - ▶▶ For a description of the set status condition, see Chapter 8, “Conditional Expressions.”
- Some database commands are affected, as listed below.

The following table lists considerations that apply to specific database commands when using native VSAM data sets:

Database commands and native VSAM data sets

Command	Consideration
ACCEPT DB-KEY RELATIVE TO CURRENCY	Next, prior, and owner currency cannot be requested for sets defined for native VSAM records.
CONNECT	The CONNECT command is not allowed because all sets in native VSAM data sets must be defined as mandatory automatic.
DISCONNECT	The DISCONNECT command is not allowed because all sets in native VSAM data sets must be defined as mandatory automatic.
ERASE	ERASE record-name is the only form of the ERASE command that is valid for use with native VSAM data sets. No form of the ERASE command is permitted against records contained in an ESDS.
FIND/OBTAIN DB-KEY	The FIND/OBTAIN DB-KEY command cannot be used to access records in a native VSAM KSDS because the database key does not necessarily remain static in a KSDS.
FIND/OBTAIN OWNER	The FIND/OBTAIN OWNER command is not allowed because owner records are not defined in native VSAM data sets.
FIND/OBTAIN WITHIN SET/AREA	When an end-of-set or end-of-area condition occurs, all currencies remain unchanged. The FIRST, LAST, and sequence-vn WITHIN AREA options cannot be used to access spanned data records in a native VSAM data set.

Command	Consideration
MODIFY	<p>The length of a record in an ESDS file cannot be changed even if the record is variable length.</p> <p>The prime key for a KSDS cannot be modified.</p>
STORE	<p>If the object record is to be stored in a native VSAM RRDS, the DIRECT-DBKEY field must be initialized with the relative record number of the record being stored.</p>

16.2.3 Record locking

Record locks are used to protect the integrity of database records.

Share and exclusive locks: Record locks protect object records from concurrent access or update by other run units. Locks can be shared or exclusive:

- **Shared record locks** allow other run units to access but not update the locked record.
- **Exclusive record locks** prohibit other run units from accessing the locked record as long as the lock is maintained.

Implicit and explicit record locks: Record locks can be set implicitly by the CA-IDMS/UCF central version and explicitly by the application developer, as follows:

- **Implicit record locks** are maintained automatically for every run unit that executes in shared update usage mode. Usage modes are discussed in 'READY' later in this section.
- **Explicit record locks** are set by means of a KEEP command or the KEEP clause of a FIND/OBTAIN command. FIND/OBTAIN is described later in this section.

Long-term explicit record locks are shared or exclusive record locks that are maintained across run units. A long-term lock placed on a record restricts other concurrently executing run units from accessing or updating the record until the lock is explicitly released. Subsequent run units in the same CA-ADS application that execute from the same terminal can access and update the locked record, and can upgrade or release the long-term lock.

►► More information about record locks can be found in *CA-IDMS Database Design*.

The following conditions resulting from the use of record locks can cause abnormal termination of a CA-ADS application:

- **Too many locks** — Abnormal termination of a CA-ADS application occurs if a run unit tries to generate more record locks than the maximum number specified

at CA-IDMS/UCF system generation. To lessen the possibility of abnormal termination because of too many locks, a COMMIT command can be used to release locks.

►► The COMMIT command is described in 16.2.11, “COMMIT” later in this section.

- **Wait time** — Abnormal termination of a CA-ADS application occurs if the internal wait time of a run unit exceeds the wait interval specified at CA-IDMS/UCF system generation.
- **Deadlock** — Abnormal termination of a run unit occurs when two run units would cause a deadlock by being permitted to wait to set locks. The run-unit that would complete the deadlock terminates, control returns to the issuing task, and a minor code 29 is returned.

An online application can include logic that is invoked if the run unit is terminated because of a db-key deadlock. In this way, the application can maintain the terminal session and save data previously entered on the screen. The application can then ask the user to resubmit the transaction or automatically restart the run unit, establish currency, and try again.

If the run unit is automatically restarted, the following steps should be followed:

1. **Rebind the run unit.** CA-ADS automatically starts a new run unit when it encounters the first functional DML statement.
2. **Reestablish currency.** If appropriate currencies are not reestablished before retrying the operation that initially caused the deadlock, a status code of *nn06* (no currency established) will be returned.

►► See *CA-IDMS Navigational DML Programming* for handling the minor code 29.

Checking for deadlock conditions: Deadlock conditions can be checked for programmatically by using the ALLOWING clause when autostatus is enabled. The check for a deadlock condition can be made after each service request to the DBMS.

►► The ALLOWING clause is discussed in Chapter 10, “Error Handling.”

More information about record contention can be found in *CA-IDMS Database Design*.

16.2.4 Suppression of record retrieval locks

Specifications can be made during dialog compilation to indicate whether or not database record retrieval locks will be held for dialog run units. Retrieval dialogs that **do not update the database** and **do not pass currencies to update dialogs** can be selectively allowed to access database records without locking those records.

Selectively disabling retrieval locks for dialogs allows:

- Elimination of the overhead of maintaining retrieval locks. This decreases the amount of potential storage and CPU time used by dialogs at runtime.
- Reduction of the number of db-key deadlocks.

Disabling record retrieval locks: To disable record retrieval locks, you must:

1. **Analyze the dialog in the context of the entire application** to ensure that control and currencies are passed appropriately. A dialog with disabled retrieval locks can pass control and currencies only to a dialog or user program that does retrieval based on these currencies.
2. **Verify the status of the system retrieval locks.** If the mandatory retrieval locks are on, disable the locks at system generation time by specifying RETRIEVAL NOLOCK in the system generation SYSTEM statement.

►► See *CA-IDMS System Generation* for additional information.

3. **Use the CA-ADS dialog compiler or ADSOBCOM** to disable retrieval locking for appropriate dialogs.

►► The dialog compiler is described in Chapter 2, “CA-ADS Application Compiler (ADSA).”

ADSOBCOM is described in Appendix D, “Application and Dialog Utilities.”

Considerations

- To safeguard the database in the absence of retrieval locks, an update user program will be aborted when:
 - The program receives currencies from a retrieval dialog and attempts an update DML call.
 - The program finishes the current run unit and binds another. The abend occurs when control is passed back to CA-ADS.
- The update dialog abends if:
 - A higher dialog in the application thread has the RETRIEVAL NOLOCK indicator set and system-wide RETRIEVAL NOLOCKS are specified.
- An update dialog or program is allowed to update the retrieval dialog's database records in the following cases:
 - The dialog with retrieval locks turned off readies the area in UPDATE mode.
 - The update dialog/program *does not* receive currencies when control passes to it.

Updates are allowed because the update dialog/program must ready the database in UPDATE mode and establish its own currency. The dialog/program will use record-locking mechanisms and will be assured of having the most up-to-date data.

The control command options that avoid passing currencies when control is passed are the TRANSFER command and the NOSAVE clause of the DISPLAY, INVOKE, and LINK commands.

►► Complete syntax for these commands is given in Chapter 15, “Control Commands.”

16.2.5 Overview of ACCEPT

The ACCEPT command moves database keys page information and statistics from the database management system to a dialog's record buffers.

Formats of the ACCEPT command: The ACCEPT command has three formats, as outlined in the table below.

Format	Description
ACCEPT DB-KEY FROM CURRENCY	Saves the database key of the current record of run unit, record type, set, or area
ACCEPT DB-KEY RELATIVE TO CURRENCY	Saves the database key of the next, prior, or owner record relative to the current record of a set
ACCEPT PAGE-INFO	Saves the page information of the record named.
ACCEPT STATISTICS	Returns runtime database statistics to the dialog

Note:

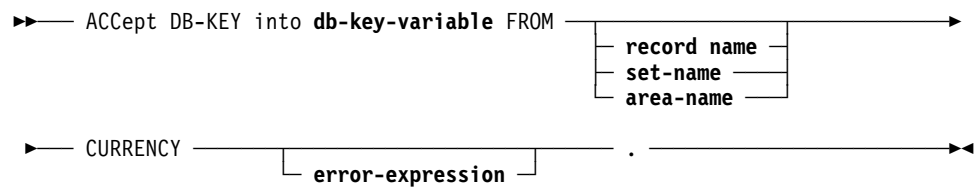
The ACCEPT utility command should not be confused with the ACCEPT database command. The ACCEPT utility command is used to access information about the current CA-IDMS/UCF task.

For a description of the ACCEPT utility command, see Chapter 20, “Utility Commands.”

16.2.6 ACCEPT DB-KEY FROM CURRENCY

Purpose: Saves the database key of the current record of run unit, record type, set, or area.

Syntax



Parameters

ACCEpt DB-KEY into db-key-variable

Specifies the variable data field to which the database key of the object record is moved.

Db-key-variable is a PIC S9(8) COMP SYNC.

Db-key-variable must be a binary fullword field that is defined in a record associated with the dialog.

FROM

Specifies the record whose database key is moved to the field identified by *db-key-variable*.

record-name

Saves the database key of the record that is current of the specified record type.

set-name

Saves the database key of the record that is current of the specified set.

area-name

Saves the database key of the record that is current of the specified area.

CURRENCY

Specifies the current record of run unit, record type, set, or area.

If no record, set, or area is specified, CA-ADS saves the database key of the record that is current of run unit.

error-expression

Specifies the status codes that are returned to the dialog.

►► Error expressions are described in Chapter 10, "Error Handling."

Usage:

Definition: The `ACCEPT DB-KEY FROM CURRENCY` command is used to move the database key of the current record of run unit, record type, set, or area to a specified location in a dialog's record buffers. A subsequent `FIND/OBTAIN DB-KEY` command can use the saved database key to access the record directly. `FIND/OBTAIN DB-KEY` is described later in this section.

Note: You must establish currency before using this statement. If no currency has been established, the DBMS returns 0000 to the `ERROR-STATUS` field and -1 to the *db-key* field.

Considerations: If autostatus is not in use, a dialog's error-status field indicates the outcome of an ACCEPT DB-KEY FROM CURRENCY command:

Status code	Meaning
0000	The request was executed successfully
1508	The object record is not in the dialog's subschema

►► The autostatus facility is described in Chapter 10, "Error Handling."

Example: The statements in the following example establish a PRODUCT record as current of run unit and save the record's database key in the field SAVE-DB-KEY:

```
MOVE 7690157 TO PROD-NUMBER.
FIND CALC PRODUCT.
ACCEPT DB-KEY INTO SAVE-DB-KEY FROM CURRENCY.
```

16.2.7 ACCEPT DB-KEY RELATIVE TO CURRENCY

Purpose: Saves the database key of the next, prior, or owner record relative to the current record of a set.

Syntax

```
►► — ACccept DB-KEY into db-key-variable FROM set-name —————►
      |
      | NEXT | ——— CURRENCY ——— [ error-expression ] ——— . —————►
      | PRIOR |
      | OWNER |
```

Parameters

ACccept DB-KEY into **db-key-variable**

Specifies the variable data field to which the database key of the object record is moved.

Db-key-variable is a PIC S9(8) COMP SYNC.

Db-key-variable must be a binary fullword field that is defined in a record associated with the dialog.

FROM **set-name**

Specifies the record whose database key is moved to the field identified by *db-key-variable*.

Set-name must be known to the dialog's subschema.

NEXT

Saves the database key of the next record relative to the current record of the specified set.

A request for NEXT CURRENCY cannot be specified unless the object set has prior pointers, which ensure that the next pointer in the prefix of the current record does not point to a logically deleted record.

PRIOR

Saves the database key of the prior record relative to the current record of the specified set.

A request for PRIOR CURRENCY cannot be specified unless the object set has prior pointers.

Note: No indication of an end-of-set condition is possible for an ACCEPT NEXT or ACCEPT PRIOR command. A retrieval command must be issued to determine whether the next or prior record in the specified set is the owner record.

OWNER

Saves the database key of the owner of the current record of the specified set.

A request for OWNER CURRENCY cannot be specified unless the object set has owner pointers. If the current record is the owner of the specified set, a request for OWNER CURRENCY returns the database key of the current record, even if the set does not have owner pointers.

CURRENCY

Specifies the current record of run unit, record type, set, or area.

error-expression

Specifies the status codes that are returned to the dialog.

►► Error expressions are described in Chapter 10, “Error Handling.”

Usage:

Definition: The ACCEPT DB-KEY RELATIVE TO CURRENCY command is used to move the database key of the next, prior, or owner record relative to the current record of set to a specified location in a dialog's record buffers.

This command allows a process to save the database key of a record without accessing the record itself. A subsequent FIND/OBTAIN DB-KEY command can use the saved database key to access the record directly. FIND/OBTAIN DB-KEY is described later in this section.

Note: You must establish currency before using this statement. If no set currency has been established, the DBMS returns 0000 to the ERROR-STATUS field and -1 to the *db-key* field.

Note: NEXT, PRIOR, and OWNER CURRENCY cannot be requested for sets defined for native VSAM records.

If autostatus is not in use, a dialog's error-status field indicates the outcome of an ACCEPT DB-KEY RELATIVE TO CURRENCY command:

Status code	Meaning
0000	The request was executed successfully
1506	Currency was not established for the object set
1508	The object record is not in the dialog's subschema

►► The autostatus facility is described in Chapter 10, “Error Handling.”

Example: The statements in the following example establish a current ITEM record and save the database key of the owner record of the PRODUCT-ITEM set in the field SAVE-KEY:

```
MOVE 1230407 TO ORD-NUMBER.
FIND CALC ORDOR.
FIND NEXT WITHIN ORDER-ITEM.
ACCEPT DB-KEY INTO SAVE-KEY FROM PRODUCT-ITEM OWNER CURRENCY.
```

16.2.8 ACCEPT PAGE-INFO

Purpose: The ACCEPT PAGE-INFO statement moves the page information for a given record to a specified location in program variable storage. Page information that is saved in this manner is available for subsequent direct access by using a FIND/OBTAIN DB-KEY statement.

The dbkey radix portion of the page information can be used in interpreting a dbkey for display purposes and in formatting a dbkey from page and line numbers. The dbkey radix represents the number of bits within a dbkey value that are reserved for the line number of a record. By default, this value is 8, meaning that up to 255 records can be stored on a single page of the area. Given a dbkey, you can separate its associated page number by dividing the dbkey by 2 raised to the power of the dbkey radix. For example, if the dbkey radix is 4, you would divide the dbkey value by $2^{**}4$. The resulting value is the page number of the dbkey. To separate the line number, you would multiply the page number by 2 raised to the power of the dbkey radix and subtract this value from the dbkey value. The result would be the line number of the dbkey. The following two formulas can be used to calculate the page and line numbers from a dbkey value:

- Page-number = dbkey value / (2 ** dbkey radix)
- Line-number = dbkey value - (page-number * (2 ** dbkey radix))

Syntax

```

▶▶ ACcEpt PAGE-INFO into page-info-variable FOR record-name
▶
└─ error-expression ─┘

```

Parameters

ACCEPT PAGE-INFO into page-info-variable

Specifies the variable data field to which the page info of the named record is moved.

page-info-variable

A four-byte field that may be defined either as a group field or as a fullword field (PIC S9(8) COMP). Identifies the variable data field to contain the page information for the specified record. Upon successful completion of this statement, the first two bytes of the field contain the page group number and the last two bytes contain a value that may be used for interpreting dbkeys.

FOR record-name*record-name*

Specifies the record whose page information will be placed in the specified location (page-info-variable).

Note: Page information is only used if the subschema includes areas that have mixed page groups; otherwise it is ignored.

Status codes: If autostatus is not in use, a dialog's error-status field indicates the outcome of an ACCEPT-PAGE-INFO command:

Status code	Meaning
0000	The request has been serviced successfully.
1508	The named record is not in the subschema. The program probably invoked the wrong subschema.

Example: The following example retrieves the page information for the DEPARTMENT record.

```
01 W-PG-INFO.
   02 W-GRP-NUM      PIC S9(4) COMP.
   02 W-DBK-FORMAT  PIC 9(4) COMP.

ACCEPT PAGE-INFO into W-PG-INFO FOR DEPARTMENT.
```

16.2.9 ACCEPT STATISTICS

Purpose: Returns runtime database statistics to the dialog.

Syntax

```
►► ACcept  ┌ STATISTICS ─┐ into db-statistics-variable ─►►
          └ STATS ───┘

► FROM IDMS-STATISTICS ─ . ─►►
```

Parameters

ACCEpt STATISTICS

Introduces the variable data field to which the database key of the object record is moved.

STATS can be used in place of STATISTICS.

into db-statistics-variable

The name of the location in the dialog's record buffers where the runtime statistics contained in the CA-IDMS statistics block are to be moved.

A fullword aligned, 100-byte system supplied statistics block shown below:

```
01 DB-STATISTICS
03 DATE-TODAY          PIC X(8).
03 TIME-TODAY          PIC X(8).
03 PAGES-READ          PIC S9(8)  COMP.
03 PAGES-WRITTEN       PIC S9(8)  COMP.
03 PAGES-REQUESTED     PIC S9(8)  COMP.
03 CALC-TARGET         PIC S9(8)  COMP.
03 CALC-OVERFLOW       PIC S9(8)  COMP.
03 VIA-TARGET          PIC S9(8)  COMP.
03 VIA-OVERFLOW        PIC S9(8)  COMP.
03 LINES-REQUESTED     PIC S9(8)  COMP.
03 RECS-CURRENT        PIC S9(8)  COMP.
03 CALLS-TO-IDMS       PIC S9(8)  COMP.
03 FRAGMENTS-STORED    PIC S9(8)  COMP.
03 RECS-RELOCATED      PIC S9(8)  COMP.
03 LOCKS-REQUESTED     PIC S9(8)  COMP.
03 SEL-LOCKS-HELD      PIC S9(8)  COMP.
03 UPD-LOCKS-HELD      PIC S9(8)  COMP.
03 RUN-UNIT-ID         PIC S9(8)  COMP.
03 TASK-ID             PIC S9(8)  COMP.
03 LOCAL-ID            PIC X(8).
03 FILLER              PIC X(8).
```

Note: Record DB-STATISTICS is defined in the dictionary when CA-IDMS is installed and can be included as a dialog work record.

►► For a description of the CA-IDMS statistics block, refer to *CA-IDMS Database Administration*.

The LOCAL-ID field consists of the 4-byte identifier of the interface in which the run unit originated (in CA-ADS, it is always DBDC) and a unique identifier (a fullword binary value) assigned to the run unit by that interface. To display the originating interface identifier and the run-unit identifier, the LOCAL-ID field can be moved to a work field that is defined as follows:

```
01 WORK-LOCAL-ID
02 WORK-LOCAL-ORIGIN   PIC X(4).
02 WORK-LOCAL-NUMBER   PIC S9(8)  COMP.
```

Alternatively, the DB-STATISTICS record can be modified to define two subordinate fields for the LOCAL-ID field.

Usage:

Definition: The ACCEPT STATISTICS command is used to move runtime statistics in the CA-IDMS statistics block to a dialog's record buffers. An ACCEPT STATISTICS command does not reset fields in the CA-IDMS statistics block. The fields are initialized at the beginning of a run unit. The only acceptable status code returned for an ACCEPT STATISTICS command is 0000.

Example: The statements in the following example:

- Establish currency for the sets in which a new ITEM record will participate as a member
- Store the ITEM record
- Move statistics regarding the stored ITEM record to the SAVE-STATS field in the dialog's record buffers

Sample Statements:

```
MOVE IN-PROD-NUMBER TO PROD-NUMBER.
FIND CALC PRODUCT.
MOVE IN-ORD-NUMBER TO ORD-NUMBER.
FIND CALC ORDOR.
STORE ITEM.
ACCEPT STATS INTO SAVE-STATS FROM IDMS-STATISTICS.
```

16.2.10 BIND PROCEDURE

Purpose: Establishes communication between a dialog and a DBA-written procedure.

Syntax:

```
►— BIND PROCedure for procedure-name TO —————►
► procedure-control-location —————┐
                                error-expression ┘ . —————►
```

Parameters

BIND PROCedure for procedure-name

Provides the name of the database procedure.

Procedure-name is either the name of an 8-character variable field that contains the procedure name or the procedure name itself enclosed in single quotation marks.

TO procedure-control-location

Specifies the location to which the named procedure is bound.

Procedure-control-location specifies a 256-byte, fixed-length area.

When the BIND PROCEDURE command is executed, information specified in the CA-IDMS application program information block is copied into *procedure-control-location*. At runtime, this information is copied from *procedure-control-location* back into the CA-IDMS application program information block each time the DBMS invokes the procedure. The information passed at runtime is not the information in storage at the time of the procedure call.

error-expression

Specifies status codes that are returned to the dialog.

►► Error expressions are described in Chapter 10, "Error Handling."

Usage:

Definition: This statement should be used when the application must pass more information to the procedure than that provided by the DBMS. Such instances are unusual.

In most cases, procedures that gain control before or after various database functions are not apparent. After the BIND PROCEDURE command is executed, the DBMS automatically invokes the named procedure for the operations specified in the schema definition.

►► See *CA-IDMS Database Administration* for more information about database procedures.

Considerations: If autostatus is not in use, a dialog's error-status field indicates the outcome of a BIND PROCEDURE command:

Status code	Meaning
0000	The request was executed successfully
1408	The named record or procedure was not in the specified subschema.
1418	The procedure was improperly bound to location 0
1472	The available memory to load a module from the load (core-image) library or DDLDCLOD was not sufficient
1474	An attempt to load a module from the load (core-image) library or DDLDCLOD failed

Example: In the example below, the BIND PROCEDURE command is used to bind the procedure PROGCHK to the 256-byte area PROC-CTL.

BIND PROCEDURE FOR 'PROGCHK' TO PROC-CTL.

16.2.11 COMMIT

Purpose: Ends the current recovery unit and makes permanent any changes made to the database data during the current recovery unit.

Syntax:

►► COMMIT TASK ALL .

Parameters**TASK**

COMMIT TASK writes a checkpoint to the CA-IDMS/DB journal file and updates the subschema control block for all database, queue, and scratch records

associated with run units that have been implicitly established for the issuing dialog. All record locks except those held on current records are released.

If TASK is not specified, only database records are the objects of the COMMIT command.

ALL

Releases all record locks, including those held on current records, and sets all currencies to null.

Note: The COMMIT command does not release long-term locks held on database records.

Usage:

Definition: The COMMIT command is used to write a checkpoint to the CA-IDMS/DB journal file and to release record locks held on database, queue, and scratch records. The checkpoints mark the beginning or end of specific database, queue, and scratch area activities within the issuing dialog. The release of record locks lessens the possibility of abnormal termination resulting from too many locks.

The CA-ADS runtime system automatically writes a checkpoint to the CA-IDMS/DB journal file at the beginning and end of a run unit. Additional checkpoints can be written to the journal file by using the COMMIT command.

►► For a discussion of CA-ADS run units, see Chapter 4, “CA-ADS Runtime System.”

Considerations: If autostatus is not in use, a dialog's error-status field indicates the outcome of a COMMIT command:

Status code	Meaning
0000	The request was executed successfully
5031	The request is invalid, possibly due to a logic error in the process
5096	Too many run units exist for the internal run-unit table
5097	An invalid status was received from DBIO. Check the CA-IDMS/UCF system log for details

►► The autostatus facility is described in Chapter 10, “Error Handling.”

16.2.12 CONNECT

Purpose: Establishes a record occurrence as a member in a set occurrence.

Participation of records in sets is governed by the membership options defined for each set in the subschema, as shown below.

Membership option	Description
Automatic	Membership is established automatically when a record is stored.
Manual	Membership is not established automatically. A record is established as a member of the set by using the CONNECT command.
Mandatory	Records remain members of the set until they are erased.
Optional	Records remain members of the set until they are erased or disconnected. For information on erasing or disconnecting a record, see 'ERASE' and 'DISCONNECT' later in this section.

Syntax:

```
➤ CONNECT record-name TO set-name [error-expression] . ➤
```

Parameters

record-name

Specifies the current occurrence of the named record to be connected with the current occurrence of the set specified by *set-name*.

Record-name must be known to the dialog's subschema.

TO set-name

Specifies the set to which the current occurrence of the named record is connected.

Set-name must be known to the dialog's subschema and must be defined as optional automatic, optional manual, or mandatory manual.

The record is connected to the current occurrence of the named set in the order specified for the set in the schema.

error-expression

Specifies the status codes that are returned to the dialog.

➤ Error expressions are described in Chapter 10, "Error Handling."

Usage:

Considerations: If autostatus is not in use, a dialog's error-status field indicates the outcome of a CONNECT command:

Status code	Meaning
0000	The request was executed successfully
0705	The CONNECT command violates a duplicates-not-allowed option for a CALC, sorted, or index set
0706	Currency was not established for the object record or set
0708	The object record is not in the dialog's subschema
0709	The object record's area was not readied in an update usage mode
0710	The dialog's subschema specifies an access restriction that prohibits connecting the object record to the named set
0714	The CONNECT command cannot be executed because the object set was defined as mandatory automatic
0716	The CONNECT command cannot be executed because the object record is already a member of the named set
0721	An area other than the area of the object record was readied with an incorrect usage mode
0729	A run-unit deadlock condition occurred. DBMS aborted and rolled back the run unit. All resources associated with the task are released

►► The autostatus facility is described in Chapter 10, “Error Handling.”

Further considerations

- The object set in a CONNECT command must be defined as optional automatic, optional manual, or mandatory manual.
- The CONNECT command cannot be used with native VSAM data sets because all such sets must be defined as mandatory automatic.
- Before a CONNECT command can be executed, the following conditions must be satisfied:
 - All areas affected either directly or indirectly by the CONNECT command must be readied in an update usage mode.
 - Usage modes are discussed in 16.2.26, “READY” later in this section.
 - The object record must be established as current of its record type.
 - The applicable set occurrence must be established by the current record of set. If set order is NEXT or PRIOR, the current record of set also determines the position at which the object record is connected within the set.

- After successful execution of the CONNECT command, the object record is current of:
 - The run unit
 - Its record type
 - Its area
 - All sets in which the record currently participates

Example: The statements in the following example establish currency for the ITEM and PRODUCT record types, and connect the current ITEM record to the set occurrence established by the current PRODUCT record:

```
MOVE 'BB' TO ORD-NUMBER.  
FIND CALC ORDOR.  
OBTAIN FIRST WITHIN ORDER-ITEM.  
MOVE ITEM-PROD-NUMBER TO PROD-NUMBER.  
FIND CALC PRODUCT.  
CONNECT ITEM TO PRODUCT-ITEM.
```

16.2.13 DISCONNECT

Purpose: Disconnects a record occurrence from a set occurrence in which it participates as a member.

Membership in the object set must be defined as OPTIONAL in the dialog's schema.

Syntax:

```
►— DISCONNECT record-name FROM set-name —┐ error-expression ┘ . —►
```

Parameters

record-name

Specifies the current occurrence of the named record to be disconnected.

Record-name must be known to the dialog's subschema.

FROM set-name

Specifies the set from which the current occurrence of the named record is to be disconnected.

Set-name must be known to the dialog's subschema and must be defined as optional.

►► Set membership options are described in 16.2.12, "CONNECT" above.

error-expression

Specifies the status codes that are returned to the dialog.

►► Error expressions are described in Chapter 10, "Error Handling."

Usage:

Considerations: If autostatus is not in use, a dialog's error-status field indicates the outcome of a DISCONNECT command:

Status code	Meaning
0000	The request was executed successfully
1106	Currency was not established for the object record
1108	The specified record is not in the dialog's subschema
1109	The object record's area was not readied in an update usage mode
1110	The dialog's subschema specifies an access restriction that prohibits disconnecting the object record from the named set
1115	The DISCONNECT command cannot be executed because the object set is defined as mandatory

Status code	Meaning
1121	An area other than the area of the object record was readied with an incorrect usage mode
1122	The object record is not currently a member of the specified set
1129	A run-unit deadlock condition occurred. DBMS aborted and rolled back the run unit. All resources associated with the task are released

►► The autostatus facility is described in Chapter 10, “Error Handling.”

Further considerations

- The DISCONNECT command cannot be used with native VSAM data sets because all such sets must be defined as mandatory automatic.
- Before a DISCONNECT command can be executed, the following conditions must be satisfied:
 - All areas affected either directly or indirectly by the DISCONNECT command must be readied in an update usage mode.
 - Usage modes are discussed in 16.2.26, “READY” later in this section.
 - The object record must be established as current of its record type.
- After successful execution of a DISCONNECT command, the object record can no longer be accessed through the set for which membership was canceled.
- A disconnected record can be accessed through any other sets in which it participates as a member or through its CALC key if it has a location mode of CALC.
- A disconnected record is always accessible by means of an area search or through its database key.
- A DISCONNECT command nullifies currency in the object set. However, the next of set and prior of set are maintained, enabling access to continue within the set.
- A disconnected record becomes current of:
 - The run unit
 - Its record type
 - Its area

Example: The statements in the following example establish an ITEM record as current of record type and disconnect the record from the PRODUCT-ITEM set:

```

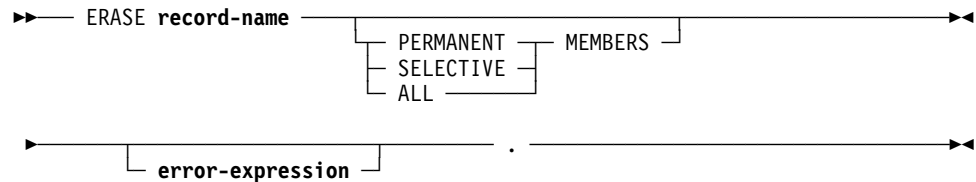
MOVE 'P8' TO PROD-NUMBER.
FIND CALC PRODUCT.
FIND FIRST ITEM WITHIN PRODUCT-ITEM.
DISCONNECT ITEM FROM PRODUCT-ITEM.

```

16.2.14 ERASE

Purpose: Deletes a record from the database.

Syntax:



Parameters

record-name

Erases the current occurrence of the named record from the database.

Record-name must be known to the dialog's subschema and must be current of run unit.

Note: Native VSAM users — ERASE *record-name* is the only form of the ERASE statement valid for records in a native VSAM KSDS or RRDS; no form of the ERASE statement is allowed for a native VSAM ESDS.

PERMANENT

Specifies the named record and all mandatory member record occurrences owned by the record to be erased. Optional member records are disconnected.

An erased mandatory member record that is itself the owner of any set occurrences is also treated as the direct object of an ERASE PERMANENT command (that is, all mandatory members in the sets owned by the record are also erased).

SELECTIVE

Specifies the named record and all mandatory member record occurrences owned by the record to be erased. Optional member records are erased if they do not currently participate as members in other set occurrences.

An erased member record that is itself the owner of any set occurrences is also treated as the direct object of an ERASE SELECTIVE command.

ALL

Specifies the named record, all mandatory and optional member record occurrences owned by the record to be erased.

An erased member record that is itself the owner of any set occurrences is also treated as the direct object of an ERASE ALL command.

MEMBERS

Must be specified if the record identified by *record-name* is the owner of any nonempty set occurrences.

error-expression

Specifies the status codes that are returned to the dialog.

►► Error expressions are described in Chapter 10, “Error Handling.”

Usage

Definition: Erasure is a two-step process that first cancels a record's membership in any set occurrences and then releases for reuse the space occupied by the record.

The ERASE command performs the following functions:

- Erases the object record from the database
- Erases all records that are mandatory members of set occurrences owned by the object record
- Disconnects or erases all records that are optional members of set occurrences owned by the object record

Considerations: If autostatus is not in use, a dialog's error-status field indicates the outcome of an ERASE command:

Status code	Meaning
0000	The request was executed successfully
0206	Currency was not established for the object record
0209	The object record's area was not readied in an update usage mode
0210	The dialog's subschema specifies an access restriction that prohibits use of the ERASE command.
0213	Run-unit currency was not established or was nullified by a previous ERASE command
0220	The current record of run unit is not the same type as the specified record
0221	An area other than the area of the object record was readied with an incorrect usage mode
0225	Currency was not established for the object record or set
0229	A run-unit deadlock condition occurred. DBMS aborted and rolled back the run unit. All resources associated with the task are released
0230	An attempt was made to erase the owner of a nonempty set

Status code	Meaning
0233	Erasure of the object record is not allowed by the dialog's subschema, or not all sets in which the object record participates are included in the subschema

►► The autostatus facility is described in Chapter 10, “Error Handling.”

Further considerations:

- Before an ERASE command can be executed, the following conditions must be satisfied:
 - All areas either directly or indirectly affected by the ERASE command must be readied in an update usage mode.
 - Usage modes are discussed in 16.2.26, “READY” later in this section.
 - All sets in which the object record participates as owner either directly or indirectly (for example, a set whose owner is a member of a set owned by the object record) and all member record types in those sets must be included in the dialog's subschema.
 - The object record must be established as current of run unit.
- An ERASE command nullifies the CURRENT pointer for all record types involved in the erase and for all sets in which erased records participate. Run-unit and area currencies remain unchanged.
- The next of set and prior of set are maintained when walking the set occurrence of an erased record, whether or not prior pointers have been defined for the sets.
- Erased records are not available for further processing. An attempt to retrieve an erased record results in an error condition.
- Next, prior, and owner pointers are preserved for the last occurrence of each record type erased. This enables access to the next or prior record within the area, or the next, prior, or owner records within the sets in which the erased record participated.

16.2.15 Overview of FIND/OBTAIN

The FIND command is used to locate a record occurrence in the database. The OBTAIN command is used to locate a record and move the data associated with the record to a dialog's record buffers. Because the FIND and OBTAIN command statements have identical formats, they are discussed together.

There are six formats of the FIND/OBTAIN statement, as outlined below.

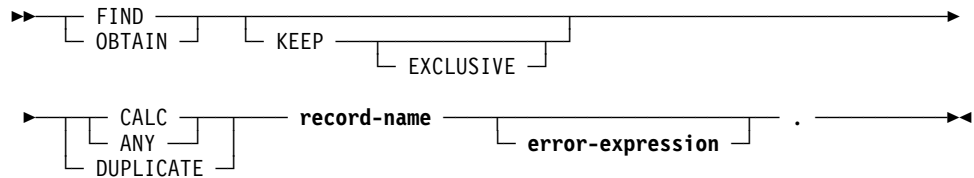
Formats of the FIND/OBTAIN statement

Format	Description
FIND/OBTAIN CALC	Accesses a record occurrence by using its CALC key value
FIND/OBTAIN CURRENT	Accesses a record occurrence by using established currencies
FIND/OBTAIN DB-KEY	Accesses a record occurrence by using its database key
FIND/OBTAIN OWNER	Accesses the owner record of a set occurrence
FIND/OBTAIN WITHIN SET/AREA	Accesses a record occurrence based on its logical location within a set or on its physical location within an area
FIND/OBTAIN WITHIN SET USING SORT KEY	Accesses a record occurrence in a sorted set by using its sort key value

Considerations

- Locks can be placed on located record occurrences by using the KEEP clause of a FIND/OBTAIN statement. The KEEP clause sets a shared or exclusive lock.
 - **KEEP** places a shared lock on the located record occurrence. Other concurrently executing run units can access but not update the locked record.
 - **KEEP EXCLUSIVE** places an exclusive lock on the located record occurrence. Other concurrently executing run units can neither access nor update the locked record.

Syntax:



KEEP

EXCLUSIVE

CALC

ANY can be used in place of CALC.

DUPLICATE

record-name

Record-name must be known to the dialog's subschema.

error-expression

Specifies the status codes that are returned to the dialog.

►► Error expressions are described in Chapter 10, “Error Handling.”

Usage:

Considerations: If autostatus is not in use, a dialog's error-status field indicates the outcome of a FIND/OBTAIN CALC command:

Status code	Meaning
0000	The request was executed successfully

Status code	Meaning
0306	Currency was not established for the object record (applies to the DUPLICATE option only)
0308	The object record is not in the dialog's subschema
0310	The dialog's subschema specifies an access restriction that prohibits retrieval of the object record
0326	The specified record cannot be found
0329	A run-unit deadlock condition occurred. DBMS aborted and rolled back the run unit. All resources associated with the task are released
0331	The object record was not defined with a location mode of CALC
0332	The value of the CALC data item in the dialog's record buffer does not equal the value of the CALC data item in the current record of run unit (applies to DUPLICATE option only)

►► The autostatus facility is described in Chapter 10, “Error Handling.”

Further considerations

- The object record must be stored in the database with a location mode of CALC.
- Before a FIND/OBTAIN CALC command is issued, the CALC key value of the object record must be placed in the applicable field of the dialog's record buffer.
- After successful execution of a FIND/OBTAIN CALC command, the accessed record is current of:
 - The run unit
 - Its record type
 - Its area
 - All sets in which it currently participates as member or owner

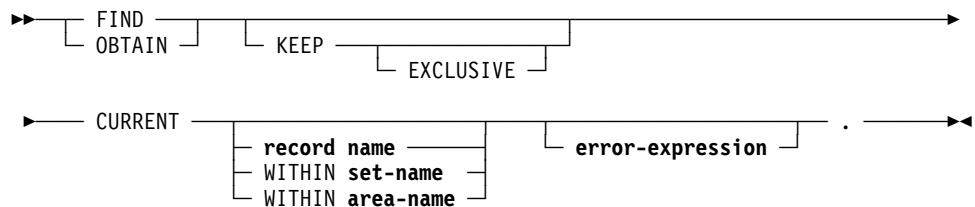
Example: The statements in the following example initialize the CALC key field in a dialog's ORDOR record buffer and retrieve the specified occurrence of the ORDOR record:

```
MOVE IN-ORDER-NUMBER TO ORD-NUMBER.  
OBTAIN CALC ORDOR.
```

16.2.17 FIND/OBTAIN CURRENT

Purpose: Accesses a record that is current of run unit, current of the record's record type or area, or current of any set in which the record participates as member or owner.

Syntax



Parameters

KEEP

Places a shared lock on the object record.

EXCLUSIVE

Places an exclusive lock on the object record.

CURRENT

Accesses the record occurrence that is current of run unit.

record-name

Specifies the current occurrence of the named record to be accessed.

WITHIN set-name

Specifies the current occurrence of the named set to be accessed.

WITHIN area-name

Specifies the current occurrence of the named area to be accessed.

The named record, set, or area must be known to the dialog's subschema.

error-expression

Specifies the status codes that are returned to the dialog.

►► Error expressions are described in Chapter 10, "Error Handling."

Usage: FIND/OBTAIN CURRENT is an efficient means of establishing a record as current of run unit before executing a command that uses run-unit currency (for example, ERASE, GET, or MODIFY).

After successful execution of a FIND/OBTAIN CURRENT command, the accessed record is current of:

- Run unit
- Record type
- Area

- All sets in which the record participates as member or owner

Considerations: If autostatus is not in use, a dialog's error-status field indicates the outcome of a FIND/OBTAIN CURRENT command:

Status code	Meaning
0000	The request was executed successfully
0306	Currency was not established for the named record, set, or area
0308	The object record is not in the dialog's subschema
0310	The dialog's subschema specifies an access restriction that prohibits retrieval of the object record
0313	Run-unit currency was not established or was nullified by a previous ERASE command
0323	The named area is not in the dialog's subschema
0329	A run-unit deadlock condition occurred. DBMS aborted and rolled back the run unit. All resources associated with the task are released

►► The autostatus facility is described in Chapter 10, "Error Handling."

Example: The statements in the following example establish an ITEM record as current of run-unit before issuing a command that requires run-unit currency:

```
MOVE 'BB' TO ORD-NUM.  
OBTAIN CALC ORDOR.  
OBTAIN FIRST ITEM WITHIN ORDER-ITEM.  
OBTAIN OWNER WITHIN PRODUCT-ITEM.  
OBTAIN CURRENT ITEM.  
MODIFY ITEM.
```

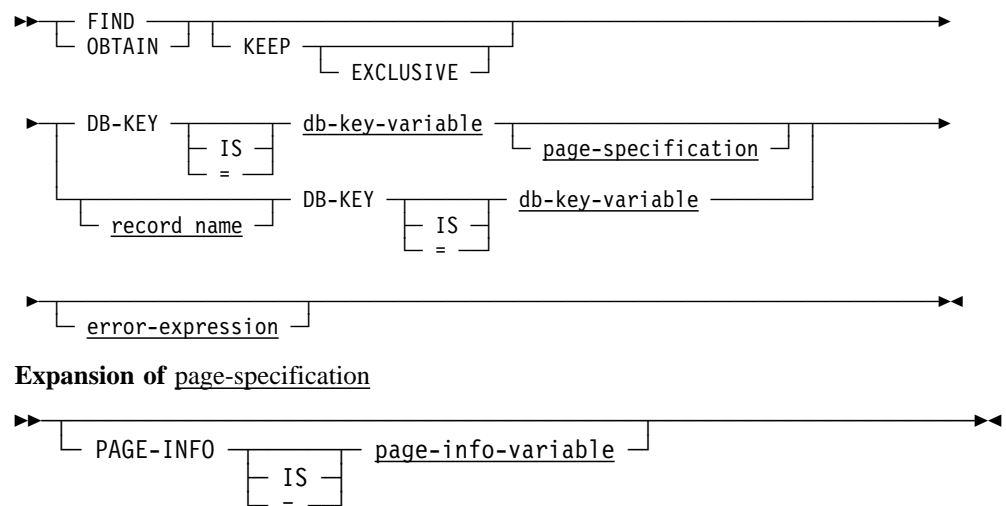
The object ITEM record becomes current of run unit following the third statement. The fourth statement establishes the owner PRODUCT record as current of run unit. The OBTAIN CURRENT statement reestablishes the ITEM record as current of run unit.

16.2.18 FIND/OBTAIN DB-KEY

Purpose: Accesses a record occurrence directly by using a database key that is stored in a field in a dialog's record buffers.

Any record in a dialog's subschema can be accessed in this manner, regardless of its location mode.

Syntax



Parameters

KEEP

Places a shared lock on the object record.

EXCLUSIVE

Places an exclusive lock on the object record.

record-name

Specifies the name of the record to be accessed using the database key value contained in *db-key-variable*.

If specified, *record-name* must be known to the dialog's subschema.

DB-KEY IS db-key-variable

Specifies the binary fullword in the dialog's record buffers that contains a previously saved database key. If *record-name* is specified, *db-key-variable* must contain the database key of an occurrence of the named record type. If *record-name* is not specified, *db-key-variable* can contain the database key of an occurrence of any record type in the dialog's subschema.

Db-key-variable is a PIC S9(8) COMP SYNC.

IS or = are optional keywords and have no effect on processing.

PAGE-INFO

Specifies page information that is used to determine the area with which the dbkey is associated. If not specified, the page information associated with the record that is current of rununit is used.

Note: Page information is only used if the subschema includes areas that have mixed page groups; otherwise, it is ignored.

page-info-variable

A four-byte field that may be defined either as a group field or as a fullword field (PIC S9(8) COMP). Identifies the location in variable storage that contains the page information previously saved by the program.

Page information is returned in the PAGE-INFO field in the subschema control area if the subschema includes areas in mixed page groups. Page information may also be returned using an ACCEPT PAGE-INFO statement.

error-expression

Specifies the status codes that are returned to the dialog.

►► Error expressions are described in Chapter 10, “Error Handling.”

Usage:

Considerations: If autostatus is not in use, a dialog's error-status field indicates the outcome of a FIND/OBTAIN DB-KEY command:

Status code	Meaning
0000	The request was executed successfully
0302	The database key value is inconsistent with the area in which the named record is stored. Either the database key was not initialized properly or the record name is incorrect
0308	The object record is not in the dialog's subschema
0310	The dialog's subschema specifies an access restriction that prohibits retrieval of the named record
0326	The specified record cannot be found
0329	A run-unit deadlock condition occurred. DBMS aborted and rolled back the run unit. All resources associated with the task are released
0371	The specified database key does not correspond to a database page

►► The autostatus facility is described in Chapter 10, “Error Handling.”

Further considerations

- FIND/OBTAIN DB-KEY cannot be used to access data records in a native VSAM KSDS.
- After successful execution of a FIND/OBTAIN DB-KEY command, the accessed record is current of the run unit, its record type, its area, and all sets in which it currently participates as member or owner.

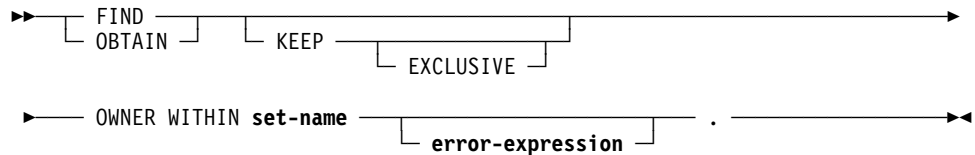
Example: The following example illustrates the use of the FIND DB-KEY command to locate an occurrence of the ITEM record whose database key matches the value in a field called SAVED-KEY:

```
FIND ITEM DB-KEY IS SAVED-KEY.
```


16.2.19 FIND/OBTAIN OWNER

Purpose: Accesses the owner record of the current occurrence of a set.

Syntax:



Parameters

KEEP

Places a shared lock on the object record.

EXCLUSIVE

Places an exclusive lock on the object record.

OWNER WITHIN set-name

Specifies the set whose owner record is to be retrieved.

Set-name must be known to the dialog's subschema.

error-expression

Specifies the status codes that are returned to the dialog.

►► Error expressions are described in Chapter 10, “Error Handling.”

Usage:

Considerations: If autostatus is not in use, a dialog's error-status field indicates the outcome of a FIND/OBTAIN OWNER command:

Status code	Meaning
0000	The request was executed successfully
0306	Currency was not established for the named record, set, or area
0308	The object record is not in the dialog's subschema
0310	The dialog's subschema specifies an access restriction that prohibits retrieval of the object record
0329	A run-unit deadlock condition occurred. DBMS aborted and rolled back the run unit. All resources associated with the task are released

►► The autostatus facility is described in Chapter 10, “Error Handling.”

Further considerations

- Currency must be established for the object set to execute a FIND/OBTAIN OWNER command.
- FIND/OBTAIN OWNER can be used to retrieve the owner record of any set in a dialog's subschema, whether or not the set has owner pointers.
- The FIND/OBTAIN OWNER command cannot be used with native VSAM data sets because owner records are not defined for such sets.
- When an optional or manual member of a set is accessed, it is not established as current of set if it is not currently connected to the object set. A subsequent attempt to access the owner record locates instead the owner of the current record of set. The IF statement can be used to determine if the accessed record is actually a member of the object set before executing the FIND/OBTAIN OWNER command.
 - ▶▶ Syntax for the IF command is presented in Chapter 14, "Conditional Commands."
- After successful execution of a FIND/OBTAIN OWNER command, the accessed record is current of:
 - The run unit
 - Its record type
 - Its area
 - All sets in which it currently participates as member or owner
- If the current record of set is the owner record when the command is executed, currency in the object set is not changed.

Example: The statements in the following example illustrate the use of the FIND/OBTAIN OWNER command:

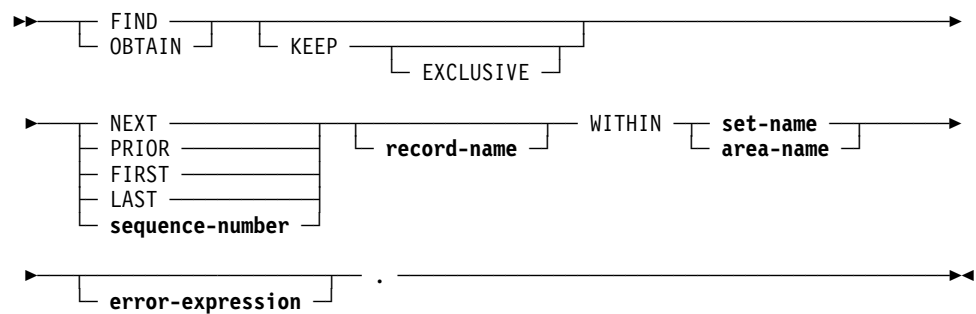
```
MOVE 'CC' TO ORD-NUM.  
OBTAIN CALC ORDOR.  
OBTAIN LAST ITEM WITHIN ORDER-ITEM.  
OBTAIN OWNER WITHIN PRODUCT-ITEM.
```

16.2.20 FIND/OBTAIN WITHIN SET/AREA

Purpose: Accesses records logically, based on set relationships, or physically, based on database location.

Records can be accessed serially in a specified set or area, or accessed by selected specific occurrences of a given record type within the set or area.

Syntax:



Parameters

KEEP

Places a shared lock on the object record.

EXCLUSIVE

Places an exclusive lock on the object record.

NEXT

Specifies the next record in the specified set or area relative to the current record.

PRIOR

Specifies the prior record in the specified set or area relative to the current record.

FIRST

Specifies the first record in the set or area.

LAST

Specifies the last record in the set or area.

sequence-number

Either the name of a variable data field that contains the sequence number of a record in a set or area or the sequence number itself expressed as a positive or negative integer. The actual sequence number, expressed as a positive or negative integer.

If *sequence-number* is negative, the specified set must have prior pointers.

record-name

Specifies only occurrences of the named record type.

Record-name must be defined as a member of the object set or be contained in the object area.

Record-name must be specified if *set-name* or *area-name* is specified, unless the record or one of the record's elements is named explicitly somewhere in the dialog's process code, or if the record is associated with the dialog as a map record.

WITHIN

Introduces the named set or area to be searched.

set-name

Specifies the set to be searched.

area-name

Specifies the area to be searched.

The named set or area must be known to the dialog's subschema.

error-expression

Specifies the status codes that are returned to the dialog.

►► Error expressions are described in Chapter 10, "Error Handling."

Usage:

Considerations: If autostatus is not in use, a dialog's error-status field indicates the outcome of a FIND/OBTAIN WITHIN SET/AREA command:

Status code	Meaning
0000	The request was executed successfully
0304	A sequence number of zero or a variable data field containing a value of zero was specified for the object record
0306	Currency was not established for the named record, set, or area
0307	The end of the set or area was reached, or the set is empty
0308	The object record is not in the dialog's subschema
0310	The dialog's subschema specifies an access restriction that prohibits retrieval of the named record
0318	The record retrieved was not bound. The record or one of the record's elements must be named explicitly somewhere in the dialog's process code, or the record must be associated with the dialog as a map record
0323	The named area is not in the dialog's subschema, or the named record is not in the named area
0326	The object record cannot be found
0329	A run-unit deadlock condition occurred. DBMS aborted and rolled back the run unit. All resources associated with the task are released

►► The autostatus facility is described in Chapter 10, "Error Handling."

Further considerations

- After successful execution of a FIND/OBTAIN WITHIN SET/AREA command, the accessed record is current of:
 - The run unit

- Its record type
 - Its area
 - All sets in which it currently participates as member or owner
- When accessing records within a set, the following considerations apply:
- The current record of the specified set determines the set occurrence to be accessed. Set currency must be established before a FIND/OBTAIN WITHIN SET command is executed.
 - The next or prior record within a set is the subsequent or previous record, relative to the current record of the named set in the logical order of the set. The prior record in a set can be retrieved only if the set has prior pointers.
 - The first or last record within a set is the first or last member occurrence, respectively, in terms of the logical order of the set. The record accessed is the same record accessed when the current of set is the owner record and the next or prior record is requested. The last record in a set can be retrieved only if the set has prior pointers.
 - The *n*th occurrence of a record within a set can be accessed by specifying a sequence number that identifies the position of the record in the set. The search begins with the owner of the current of set for the specified set and continues until the *n*th record is located or an end-of-set condition is encountered. If the specified sequence number is negative, the search proceeds in the prior direction in the set. A negative number can be specified only if the set has prior pointers.
 - When an end-of-set condition occurs, the owner record occurrence of the set becomes current of:
 - The run unit
 - Its record type
 - Its area
- The owner record also becomes current of the set named in the FIND/OBTAIN command. Currency of other sets in which the record participates as member or owner is not changed.
- If OBTAIN is specified and an end-of-set condition occurs, the contents of the owner record are not moved to the dialog's record buffer (that is, OBTAIN is treated as FIND).
- When accessing records within an area, the following considerations apply:
- The first record occurrence within an area is the one with the lowest database key. The last record occurrence is the one with the highest database key.
 - The next record within an area is the one with the next higher database key relative to the current record of the object area. The prior record is the one with the next lower database key relative to the current of area.

- The first or last record or the *n*th occurrence of a record in an area must be accessed to establish correct starting position before next or prior records are requested.

■ When using a native VSAM set, the following considerations apply:

- When an end-of-set or end-of-area condition (0307) occurs for a native VSAM set, all currencies remain unchanged.
- The FIRST, LAST, and *sequence-number* WITHIN *area-name* options cannot be used to access spanned data records in a native VSAM data set.

Example: The statements in the following example illustrate the use of the FIND/OBTAIN WITHIN SET command to retrieve records in an occurrence of the ORDER-ITEM set:

```
MOVE 'BB' TO ORD-NUM.
FIND CALC ORDOR.
OBTAIN FIRST ITEM WITHIN ORDER-ITEM.
OBTAIN NEXT WITHIN ORDER-ITEM.
OBTAIN 5 WITHIN ORDER-ITEM.
OBTAIN NEXT WITHIN ORDER-ITEM.
```

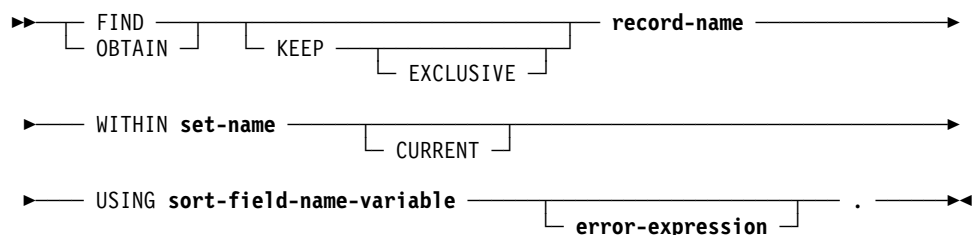
If the fifth ITEM record is the last record in the ORDER-ITEM set, the fourth OBTAIN statement finds the owner ORDOR record.

16.2.21 FIND/OBTAIN WITHIN SET USING SORT KEY

Purpose: Accesses a member record in a sorted set.

Sets are sorted in ascending or descending order based on the value of a sort-control element in each member record. The search begins with the current of set or the owner of the current of set and always proceeds through the set in the next direction.

Syntax:



Parameters

KEEP

Places a shared lock on the object record.

EXCLUSIVE

Places an exclusive lock on the object record.

record-name

Specifies the record to be accessed.

Record-name must be known to the dialog's subschema and must participate in the set specified by *set-name*.

WITHIN set-name

Specifies the set in which the object record participates.

Set-name must be known to the dialog's subschema.

CURRENT

Specifies that the search begins with the current record of the named set.

If the set is sorted in ascending order and the sort key value of the record that is current of set is higher than the sort key value specified by *sort-field-name-variable*, an error condition results. If the set is sorted in descending order and the sort key value of the record that is current of set is lower than the sort key value specified by *sort-field-name-variable*, an error condition results.

If CURRENT is not specified, the search begins with the owner of the current record of the named set.

USING sort-field-name-variable

Specifies the sort-control element to be used in searching the sorted set.

Sort-field-name-variable is either the name of the sort-control element in the record specified by *record-name* or the name of a variable data field that contains the sort key value.

error-expression

Specifies the status codes that are returned to the dialog.

►► Error expressions are described in Chapter 10, "Error Handling."

Usage:

Considerations: If autostatus is not in use, a dialog's error-status field indicates the outcome of a FIND/OBTAIN WITHIN SET USING SORT KEY command:

Status code	Meaning
0000	The request was executed successfully
0306	Currency was not established for the named set
0308	The named record or set is not in the dialog's subschema
0310	The dialog's subschema specifies an access restriction that prohibits retrieval of the object record
0326	The specified record cannot be found

Status code	Meaning
0329	A run-unit deadlock condition occurred. DBMS aborted and rolled back the run unit. All resources associated with the task are released.
0331	No sort-control element is defined for the object record in the dialog's subschema

►► The autostatus facility is described in Chapter 10, “Error Handling.”

Further considerations

- Before issuing a FIND/OBTAIN WITHIN SET USING SORT KEY command, the application developer must place the sort key value of the object record in the applicable field of the dialog's record buffer. If more than one record occurrence has a sort key value equal to the value in the record buffer, the first such record is accessed.
- After successful execution of a FIND/OBTAIN WITHIN SET USING SORT KEY command, the accessed record is current of:
 - The run unit
 - Its record type
 - Its area
 - All sets in which it currently participates as member or owner
- If the object record is not found, next of set and prior of set are maintained, but current of set is nullified.
- Next of set points to the next higher (for ascending sets) or next lower (for descending sets) sort key value.

Example: The statements in the following example establish a current PRODUCT-ITEM set and then retrieve an ITEM record based on the lot number:

```
MOVE 'P8' TO PROD-NUMBER.  
FIND CALC PRODUCT.  
MOVE 1230427 TO ITEM-LOT-NUMBER.  
FIND ITEM WITHIN PRODUCT-ITEM USING ITEM-LOT-NUMBER.
```

16.2.22 GET

Transfers the contents of a record occurrence to a dialog's record buffer.

Elements in the object record are moved to the buffer according to the subschema view of the record.

Syntax:

► GET ———┬── **record-name** ───┬── **error-expression** ───┬── . ───►

Parameters

record-name

Retrieves the record that is current of run unit. If *record-name* is specified, current of run unit must be an occurrence of the named record type.

Record-name must be specified, unless the record or one of the record's elements is named explicitly somewhere in the dialog's process code, or the record is associated with the dialog as a map record.

error-expression

Specifies the status codes that are returned to the dialog.

►► Error expressions are described in Chapter 10, “Error Handling.”

Usage:

Considerations: If autostatus is not in use, a dialog's error-status field indicates the outcome of a GET command:

Status code	Meaning
0000	The request was executed successfully
0508	The object record is not in the dialog's subschema
0510	The dialog's subschema specifies an access restriction that prohibits retrieval of the object record
0513	Run-unit currency was not established or was nullified by a previous ERASE command
0518	The record retrieved was not bound. The record or one of the record's elements must be named explicitly somewhere in the dialog's process code or the record must be associated with the dialog as a map record
0520	The current record of run unit is not the same type as the named record

►► The autostatus facility is described in Chapter 10, “Error Handling.”

Further considerations

- The GET command operates only on the record that is current of run unit.
- After the successful execution of a GET command, the accessed record remains current of run unit and becomes current of its record type, its area, and all sets in which it participates as member or owner.

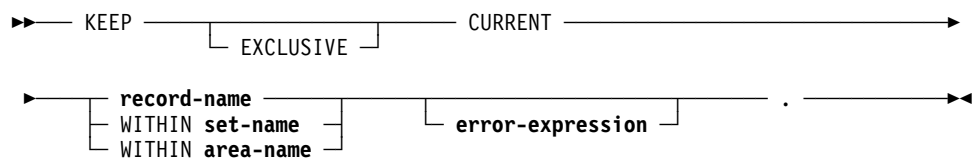
Example: The following example illustrates the use of the GET command to move the CUSTOMER record that is current of run unit to the dialog's record buffer:

GET CUSTOMER.

16.2.23 KEEP

Purpose: Places a shared or exclusive lock on a record occurrence that is current of run unit, record, set, or area.

Syntax:



Parameters

EXCLUSIVE

Places an exclusive lock on the object record. If EXCLUSIVE is not specified, the object record receives a shared lock.

CURRENT

Places a lock on the current record of run unit.

record-name

Places the record lock on the current record of the named record type.

WITHIN set-name

Places the record lock on the current record of the named set.

WITHIN area-name

Places the record lock on the current record of the named area.

error-expression

Specifies the status codes that are returned to the dialog.

►► Error expressions are described in Chapter 10, “Error Handling.”

Usage:

Definition: Record locks set with the KEEP command are maintained only for the duration of the run unit or until explicitly released by means of a COMMIT command. The COMMIT command is described earlier in this section.

Considerations: If autostatus is not in use, a dialog's error-status field indicates the outcome of a KEEP command:

Status code	Meaning
0000	The request was executed successfully

Status code	Meaning
0606	Currency was not established for the named record, set, or area
0610	The dialog's subschema specifies a privacy lock that prohibits execution of the KEEP command
0629	Deadlock occurred during locking of target record.

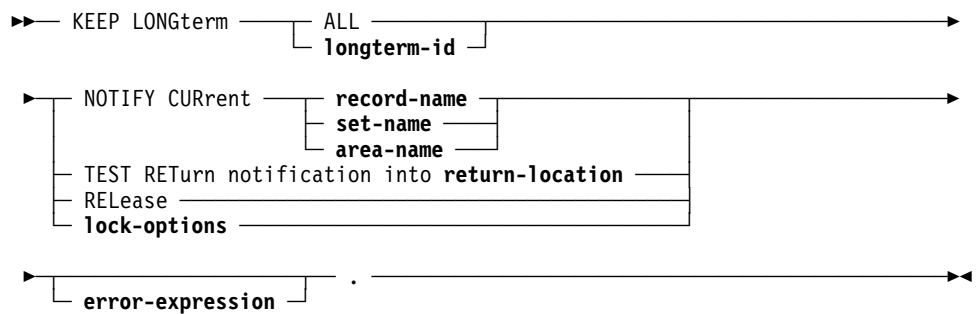
►► The autostatus facility is described in Chapter 10, “Error Handling.”

16.2.24 KEEP LONGTERM

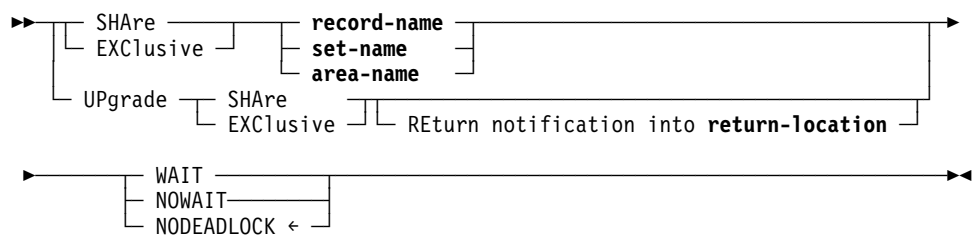
Purpose: Sets or releases long-term record locks, and monitors database activity across run units.

Information on database activity can be returned to a specified location in a dialog's record buffers.

Syntax:



Expansion of lock-options



Parameters

ALL

(Only with the RELEASE parameter) Requests release of all long-term locks associated with the current task.

longterm-id

Either the name of a 1- to 16-character variable EBCDIC data field that contains a lock identifier or the 1- to 16-character identifier itself, enclosed in single quotation marks.

Longterm-id can be used by a subsequent KEEP LONGTERM command to upgrade or release the long-term lock or to inquire about the status of database activity for the object record.

NOTIFY CURrent

Initializes a preallocated area in the dialog's record buffer with the information written by CA-IDMS/UCF on the database activity for the record identified by *longterm-id*.

record-name

Specifies monitoring of database activity for the record that is current of record type.

set-name

Specifies monitoring of database activity for the record that is current of set.

area-name

Specifies monitoring of database activity for the record that is current of area.

TEST RETurn notification into

Requests that information on database activity for the record identified by *longterm-id* be returned to the location in the dialog's record buffers specified by *return-location*.

In order to specify RETURN NOTIFICATION, a previous KEEP LONGTERM command must have included the NOTIFY CURRENT option.

return-location

The name of a binary fullword variable data field.

RELease

Releases either the long-term lock for the record identified by *longterm-id* or all long-term record locks associated with the current task.

All long-term locks that have not been released by the time the application terminates are released when the user signs off from DC/UCF with a BYE, SIGNON, or SIGNOFF command.

lock-options

Identifies the lock options.

Expanded syntax for lock-options is shown above immediately following the KEEP LONGTERM syntax.

error-expression

Specifies the status codes that are returned to the dialog.

►► Error expressions are described in Chapter 10, "Error Handling."

SHAre

Places a long-term shared lock on the object record.

EXCLUSIVE

Places a long-term exclusive lock on the object record.

Note: The shared or exclusive lock is placed only if the area in which the record is located is readied in an update usage mode.

UPGRADE

Upgrades a longterm lock placed on the record identified by *longterm-id* during execution of a previous process.

Return notification into

Clause requesting that information on database activity for the record identified by *longterm-id* be returned to the location in the dialog's record buffers specified by *return-location*.

WAIT

(Applies only to SHARE/EXCLUSIVE and UPGRADE) Places the run unit in a wait state if the lock cannot be placed immediately due to an existing lock on the record.

If waiting causes a deadlock, the requesting run unit terminates abnormally.

NOWAIT

(Applies only to SHARE/EXCLUSIVE and UPGRADE) Does not place the run unit in a wait state if the lock cannot be placed immediately due to an existing lock on the record. Control returns to the requesting run unit. The KEEP LONGTERM request is not executed.

NODEADLOCK

(Applies only to SHARE/EXCLUSIVE and UPGRADE) Places the run unit in a wait state. If waiting causes a deadlock, control returns to the requesting run unit and the KEEP LONGTERM request is not executed.

NODEADLOCK is the default when neither WAIT, NOWAIT, or NODEADLOCK is specified.

If WAIT or NODEADLOCK is specified, the run unit waits to place the lock only if the following conditions apply:

- The object record already holds an exclusive lock assigned by a concurrently executing run unit.
- The run unit that assigned the existing lock has readied the area in which the object record is located in an update usage mode.
- The run unit issuing the KEEP LONGTERM request has readied the area in which the object record is located in an update usage mode.

Control returns to the requesting run unit unless all of the above conditions apply.

Usage:

Considerations: If autostatus is not in use, a dialog's error-status field indicates the outcome of a KEEP LONGTERM command:

Status code	Meaning
0000	The request was executed successfully
0032	One of the following conditions has occurred: <ul style="list-style-type: none"> the lock id is already in use by the lterm, with a different page group or line index format a #getstg request has failed for a lock control block
0036	A lock manager error has occurred. Check the CV log for other error messages.
0044	A DCL1 error has occurred. Check the CV log for other error messages.
5101	NODEADLOCK was specified in the KEEP LONGTERM request and a deadlock condition occurred
5105	The requested record cannot be found or currency was not established for the object record
5121	Either of the following conditions has occurred: <ul style="list-style-type: none"> The requested long-term id cannot be found A KEEP LONGTERM request has been issued by a nonterminal task
5123	Area not found.
5131	Invalid param list.
5147	Area has not been readied.
5148	Run unit has not been bound.
5149	NOWAIT was specified in the KEEP LONGTERM request and a wait is required.

►► The autostatus facility is described in Chapter 10, "Error Handling."

Further considerations: When the database performs an action on an object record, one of five bit flags is turned on by the runtime system. If no bit is turned on, no database activity has occurred. The following table shows the bit assignments, their corresponding hexadecimal and decimal values, and the database activity they represent.

Bit assignment	Hexadecimal value	Decimal value	Database action
Fifth bit	X'10'	16	The record was physically deleted.
Fourth bit	X'08'	8	The record was logically deleted.
Third bit	X'04'	4	The record's prefix was modified; that is, a set operation occurred involving the record (for example, CONNECT, DISCONNECT).
Second bit	X'02'	2	The record's data was modified.
First bit	X'01'	1	The record was obtained.

Information about database activity that occurred for an object record is returned to a dialog as a decimal value. The action or combination of database actions taken can be determined by comparing the returned decimal value listed above to a constant. For example:

- If the returned value is 0, no database activity occurred for the record.
- If the returned value is 2, the record's data was modified.
- If the returned value is 6, both the record's data and the record's prefix were modified.
- If the returned value is 8 or greater, the record was deleted.
- If the returned value is 31 (the maximum possible value), all of the above actions occurred for the object record.

Examples: The following examples illustrate the use of KEEP LONGTERM to set locks and to monitor database activity:

Example 1: Setting and upgrading a lock

The following example illustrates the use of KEEP LONGTERM to set an exclusive lock on the current CUSTOMER record in one process and then to upgrade the lock to shared after the record is modified in a subsequent process:

Process A

·
·
·

```
KEEP LONGTERM LOCK-ID EXCLUSIVE CUSTOMER.
```

```
.  
.  
.
```

Process B

```
.  
.  
.
```

```
MODIFY CUSTOMER.  
KEEP LONGTERM LOCK-ID UPGRADE SHARE.
```

```
.  
.  
.
```

By upgrading the lock to shared, other concurrently executing run units are allowed to access the CUSTOMER record after it has been modified.

Example 2: Monitoring database activity

The following example illustrates the use of KEEP LONGTERM to request monitoring of database activity for the current CUSTOMER record in one process and then, in a subsequent process, to test whether the record was deleted:

Process A

```
.  
.  
.
```

```
OBTAIN CALC CUSTOMER.  
KEEP LONGTERM LOCK-ID NOTIFY CURRENT CUSTOMER.
```

```
.  
.  
.
```

Process B

```
.  
.  
.
```

```
KEEP LONGTERM LOCK-ID TEST RETURN NOTIFICATION INTO DB-ACTIV.  
IF DB-ACTIV GE 8  
THEN  
  INVOKE 'ORDCHECK'.
```

```
.  
.  
.
```


16.2.25 MODIFY

Purpose: Replaces element values of a record occurrence in the database with new element values defined in the dialog's record buffer.

Syntax:

```
►► MODIFY record-name ───────────┬── error-expression ─┬── . ───────────►◄
```

Parameters

record-name

Specifies the current occurrence of the named record to be modified with the values in the dialog's record buffer.

The named record must be known to the dialog's subschema.

error-expression

Specifies the status codes that are returned to the dialog.

►► Error expressions are described in Chapter 10, "Error Handling."

Usage:

Considerations: If autostatus is not in use, a dialog's error-status field indicates the outcome of a MODIFY command:

Status code	Meaning
0000	The request was executed successfully
0805	Modification of the record violates a duplicates-not-allowed specification for a CALC record, sorted set, or index set
0806	Currency was not established for the object set
0809	The object record's area was not readied in an update usage mode
0810	The dialog's subschema specifies an access restriction that prohibits modification of the named record
0813	Run-unit currency was not established or was nullified by an ERASE command
0820	The current record of run unit is not the same type as the named record
0821	An area other than the area of the object record was readied with an incorrect usage mode
0825	No current record of set type was established

Status code	Meaning
0829	A run-unit deadlock condition occurred. DBMS aborted and rolled back the run unit. All resources associated with the task are released
0833	Not all sorted sets in which the object record participates are included in the dialog's subschema
0855	An invalid length was defined for a variable-length record
0883	The length of a record in a native VSAM ESDS was changed or a prime key in a native VSAM KSDS was modified

►► The autostatus facility is described in Chapter 10, “Error Handling.”

Further considerations

- The following conditions must be satisfied before a MODIFY command is executed:
 - All areas affected either directly or indirectly by the MODIFY command must be readied in an update usage mode.
 - Usage modes are discussed in 16.2.26, “READY” later in this section.
 - The values of all elements defined for the object record in the dialog's subschema must be in the dialog's record buffer. If the MODIFY command is not preceded by an OBTAIN or GET command, the application developer must initialize the applicable values.
 - The object record must be established as current of run unit.
- After successful execution of a MODIFY command, the modified record is current of:
 - The run unit
 - Its record type
 - Its area
 - All sets in which it participates as member or owner
- The following special considerations apply to the modification of CALC and sort-control elements:
 - If the modification of a CALC or sort-control element violates a duplicates-not-allowed specification in the dialog's schema, the MODIFY command is not executed and an error condition results.
 - When a CALC-control element is modified successfully, the object record can be accessed by using its new CALC key value. The database key of the object record does not change.

- If a sort-control element is to be modified, the sorted set in which the object record participates must be included in the dialog's subschema.
- When a sort-control element is modified successfully, any set occurrence in which the object record currently participates as a member is examined. If necessary, the object record is disconnected and reconnected in the set occurrence to maintain the sorted set order.
- The following special considerations apply to the modification of records in native VSAM data sets:
 - The length of a record in an ESDS cannot be changed even if the records are variable length.
 - The prime key of a KSDS cannot be modified.

Example: The statements in the following example retrieve an occurrence of the CUSTOMER record by using its CALC key, update the value of the CUST-NAME element in the dialog's record buffer, and then modify the record occurrence in the database:

```
MOVE IN-CUST-NUMBER TO CUST-NUMBER.
OBTAIN CALC CUSTOMER.
MOVE NEW-CUST-NAME TO CUST-NAME.
MODIFY CUSTOMER.
```

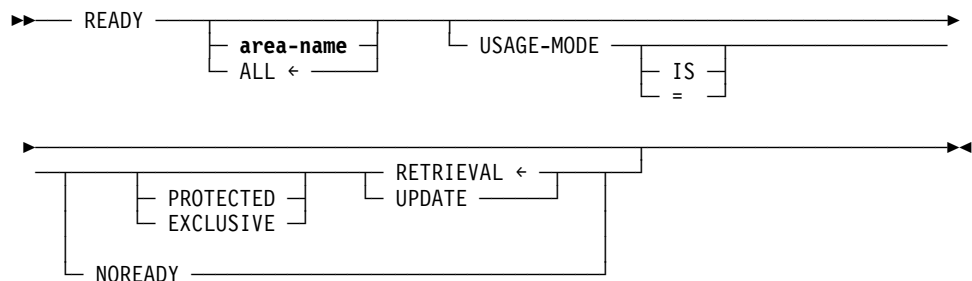
16.2.26 READY

Purpose: Overrides the usage mode specified in a dialog's subschema for one or more database areas.

Database areas are readied when a run unit begins (that is, immediately before the execution of the first database-access command issued by a process).

►► For information on CA-ADS run units, see Chapter 4, “CA-ADS Runtime System.”

Syntax:



Parameters

area-name

Readies the named area in the specified usage mode.

ALL

Readies all areas in the specified usage mode.

If neither *area-name* nor ALL is specified, all areas defined in the dialog's subschema are readied in the usage mode specified in the subschema.

USAGE-MODE

Specifies the usage mode in which the object areas are readied.

IS or = are optional keywords and have no effect on processing.

PROTECTED

Prevents concurrent update of the object areas.

EXCLUSIVE

Prevents concurrent use of the object areas.

If neither PROTECTED nor EXCLUSIVE is specified, the usage mode is qualified as shared.

RETRIEVAL

Readies the object areas for retrieval only.

RETRIEVAL is the default when neither RETRIEVAL or UPDATE is specified.

UPDATE

Readies the object areas for both retrieval and update.

NOREADY

Indicates that the area or areas named are not to be readied.

Usage

Overview of usage modes: Usage modes restrict runtime operations. Database areas can be readied in a retrieval or update usage mode.

- When an area is readied in a **retrieval usage mode**, the run unit cannot issue CONNECT, DISCONNECT, ERASE, MODIFY, or STORE commands for records in the area.
- When an area is readied in an **update usage mode**, the run unit can issue all database commands for records in the area.

Usage modes can be qualified as protected, exclusive, or shared.

- A database area readied in a **protected** retrieval or update usage mode cannot be updated by a concurrently executing run unit. A run unit cannot ready a database area in a protected usage mode if another run unit has readied the same area in an update usage mode.
- A database area readied in an **exclusive** retrieval or update usage mode cannot be used in any way by a concurrently executing run unit. A run unit cannot ready a database area in an exclusive usage mode if another run unit has readied the area in any usage mode.

- ## Considerations

- ### Example

Example 2: Specifying that an area not be readied: The following example illustrates the use of the NOREADY option. In this example, the area CUSTOMER-REGION is readied in shared update while the area ORDOR-REGION is not readied.

```
READY CUSTOMER-REGION
USAGE-MODE IS SHARED UPDATE.
READY ORDOR-REGION
USAGE-MODE IS NOREADY.
```

Syntax:

Chapter 16. Database Access Commands 16-57

Parameters**RETURN DB-KEY into**

Clause introducing the return of the database key, to the record associated with the specified index entry, to the location identified by *db-key-variable*.

db-key-variable

A numeric variable data field in the dialog's record buffers that can hold a binary fullword value.

Db-key-variable is a PIC S9(8) COMP SYNC.

FROM index-set-name

Specifies the index set associated with the index entry being retrieved.

Note: *Index-set-name* must be known to the dialog's subschema.

CURRENCY

Retrieves the entry that is current of index.

FIRST currency

Retrieves the first entry in the index.

LAST currency

Retrieves the last entry in the index.

NEXT currency

Retrieves the entry following the current of index.

PRIOR currency

Retrieves the entry preceding the current of index.

USING index-key-variable

Retrieves the first index entry whose symbolic key matches the contents of *index-key-variable*.

Index-key-variable is the name of a variable data field that is not more than 256 bytes in length.

error-expression

Specifies the status codes that are returned to the dialog.

►► Error expressions are described in Chapter 10, "Error Handling."

Usage: The RETURN DB-KEY command is used as follows:

- The value of the symbolic key in the index entry is returned to the symbolic key field for the associated record in the dialog's record buffer.
- The database key that points to the record associated with the index entry is returned to a specified field in the dialog's record buffers.
- The record referenced in the RETURN DB-KEY command must be referenced in a prior DML call in the dialog's run unit.

Considerations: If autostatus is not in use, a dialog's error-status field indicates the outcome of a RETURN DB-KEY command:

Status code	Meaning
0000	The request was executed successfully
1707	The end of the index was reached. Currency is set on the index owner. The DBMS returns the owner's db-key When 1707 is returned for an SPF index, currency remains on the last entry of the index. No db-key is returned.
1725	Index currency was not established with an OBTAIN command
1726	The index entry cannot be found
1729	A run-unit deadlock condition occurred. DBMS aborted and rolled back the run unit. All resources associated with the task are released

►► The autostatus facility is described in Chapter 10, “Error Handling.”

Further considerations

- After successful execution of a RETURN DB-KEY command, the retrieved index entry is current of index.
- If an end-of-set condition is encountered, currency is set to the last or first entry in the index, based on whether next or prior currency has been requested.
- If a specified entry cannot be found, currency is set between the two entries that are higher and lower than the specified value.
- If the specified value is higher or lower than all index entries, currency is set after or before the highest or lowest entry in the index.

16.2.28 ROLLBACK

Purpose: Requests recovery of the part of a run unit that falls between two checkpoints (a recovery unit).

Syntax

►► ROLLBACK TASK CONTINUE . ►►

Parameters

TASK

Specifies that database, queue, and scratch areas are recovered.

If TASK is not specified, only database areas are recovered.

CONTINUE

Rolls back the issuing run unit (ROLLBACK CONTINUE) or all run units associated with the issuing task (ROLLBACK TASK CONTINUE), but does not

terminate the run unit. Database access can be resumed without issuing BIND and READY statements.

Usage:

Definition: ROLLBACK performs the following functions:

- Writes an ABRT checkpoint to the CA-IDMS/DB journal file.
- Nullifies all currencies.
- Terminates database activities within the process and, if no database commands are issued after the ROLLBACK command, relinquishes control over database areas. If other database commands are issued after the ROLLBACK command, the database areas are readied again automatically in the applicable usage modes.

Considerations After successful execution of a ROLLBACK command, database, queue, and scratch areas are restored to the most recent checkpoint.: The only acceptable status code returned for a ROLLBACK command is 0000.

16.2.29 STORE

Purpose: Adds records to the database.

Syntax:

►►—— STORE **record-name---** error-expression . —————►◄

Parameters**record-name**

Specifies the name of the object record occurrence to be moved from the dialog's record buffer to the database.

Record-name must be known to the dialog's subschema.

error-expression

Specifies the status codes that are returned to the dialog.

►► Error expressions are described in Chapter 10, "Error Handling."

Usage:

Definition: The STORE command moves the object record occurrence from the dialog's record buffer to the database and connects it to an occurrence of each set for which the record is defined as an automatic member. The STORE command performs the following functions:

- Acquires space in the database and a database key for a new record occurrence
- Transfers the values of the record elements from the dialog's record buffer to the object record occurrence in the database

- Connects the object record to all sets for which it is defined as an automatic member

Considerations: If autostatus is not in use, a dialog's error-status field indicates the outcome of a STORE command:

Status code	Meaning
0000	The request was executed successfully
1202	The suggested DIRECT-DBKEY value is not within the page range for the object record
1205	Storage of the record violates a duplicates-not-allowed specification for a CALC record, sorted set, or index set
1208	The object record is not in the dialog's subschema
1209	The object record's area was not readied in an update usage mode
1210	The dialog's subschema specifies an access restriction that prohibits storage of the named record
1211	The object record cannot be stored because of insufficient space
1212	The record cannot be stored because no database key is available
1221	An area other than the area of the object record was readied with an incorrect usage mode
1225	A current of set was not established for each set to which the object record is to be connected
1229	A run-unit deadlock condition occurred. DBMS aborted and rolled back the run unit. All resources associated with the task are released.
1233	Not all sets in which the object record participates as an automatic member are included in the dialog's subschema.
1255	An invalid length was defined for a variable-length record.
1261	The record cannot be stored because of broken chains in the database.
1287	The owner and member records for a set to be updated are not in the same page group or do not have the same dbkey radix point.

►► The autostatus facility is described in Chapter 10, "Error Handling."

Further considerations

- A record occurrence is stored in the database based on the location mode specified in the schema definition of the record:

- **CALC** — The object record is placed on or near a database page that is calculated by CA-IDMS from a control element (the CALC key) in the record.
 - **VIA** — The object record is placed as close as possible to its owner record occurrence if owner and member record occurrences share a common database page range. If owner and member record occurrences do not share a common page range, the object record is placed in the same relative position in its own page range as that in which the owner record is placed in its page range.
 - **DIRECT** — The object record is placed on or near a database page that is identified by a value moved by the application developer to the DIRECT-DBKEY field.
- Before a STORE command can be executed, the following conditions must be satisfied:
- All areas affected either directly or indirectly by the STORE command must be readied in an update usage mode.
 - ▶ Usage modes are discussed in 16.2.26, “READY” earlier in this section.
 - All control elements (that is, CALC and sort keys) must be initialized.
 - If the object record has a location mode of DIRECT, the DIRECT-DBKEY field must be initialized with a suggested database key value or a null database key value of -1.
 - If the object record is to be stored in a native VSAM RRDS, the DIRECT-DBKEY field must be initialized with the relative record number that represents the location within the data set where the record is to be stored.
 - All sets in which the object record is defined as an automatic member and the owner record of each of those sets must be included in the dialog's subschema.
 - If the object record has a location mode of VIA, currency must be established for the owner of the set through which the record is stored, regardless of whether the record is an automatic or manual member of the set.
 - Currency must be established for all set occurrences for which the object record is defined as an automatic member. A STORE command connects the object record to a set occurrence, based on set order, as follows:
 - If the object record is defined as a member of a set that is ordered FIRST, the object record is connected right after the owner to become the first member of the set. If the set is ordered LAST, the object record is connected as the last member of the set.
 - If the object record is defined as a member of a set that is ordered NEXT or PRIOR, the record that is current of set establishes the set occurrence to which the object record is connected and determines the record's position within the set.

- If the object record is defined as a member of a sorted set, the process must establish currency on the set by getting currency on the set's owner. Then, the process can store the object record. CA-IDMS/DB automatically connects the object record to the correct position in the set in order to maintain the proper set sequence.

The sort key of the object record is compared with the sort key of the record that is current of set to determine if the object record can be inserted in the set by movement in the next direction. If it can, current of set remains unchanged and the object record is connected. If it cannot, current of set is repositioned at the owner record occurrence (not necessarily the current occurrence of the owner record type) and movement proceeds in the next direction until the object record can be properly connected.

- After successful execution of a STORE command, the object record becomes current of:
 - The run unit
 - Its record type
 - Its area
 - All sets in which it participates as owner or automatic member

Example: The statements in the following example store a new ITEM record in the database and connect it to the correct occurrences of the ORDER-ITEM and PRODUCT-ITEM sets:

```
MOVE IN-PROD-NUMBER TO PROD-NUMBER.  
FIND CALC PRODUCT.  
MOVE IN-ORD-NUMBER TO ORD-NUMBER.  
FIND CALC ORDOR.  
STORE ITEM.
```

16.3 Logical Record Facility commands

In CA-ADS, Logical Record Facility (LRF) commands are used to retrieve and update data that is defined in a Logical Record Facility subschema.

16.3.1 Overview of LRF database access

To enable use of LRF, DBAs predefine the paths that a dialog can use to access specific views of data in the database. Logic to navigate the database is contained in the path definition.

Given the dialog's data requirements, the programmer selects the appropriate LRF path and then codes database requests in the form of LRF commands within the dialog's process logic. At runtime, CA-IDMS/DB locates the requested data using the specified path.

Components of LRF: LRF processes commands associated with logical records. When a dialog issues an LRF command, LRF selects an appropriate path based on the information in the command statement. LRF uses field values in the record buffer that is established for the logical record to update the database.

Logical records are defined in a subschema by the database administrator (DBA). Each logical record is composed of fields selected from one or more subschema records or roles that are typically accessed together.

Logical Record Facility paths are also defined in the subschema. Each path is a group of database access instructions that perform the processing necessary to satisfy an LRF request. One or more paths are associated with each logical record in the subschema.

The predefined **conditions affecting logical record access** include:

- **Restrictions** on the commands that can be issued for each logical record
- **Selection** criteria that can be specified by a WHERE clause in each command for each logical record
- **Path statuses** returned by LRF to indicate the result of each command

To use Logical Record Facility commands effectively, the application developer must be familiar with the processing characteristics of the logical records that are defined in the subschema.

►► For more detailed information on using Logical Record Facility, refer to *CA-IDMS Logical Record Facility*.

Process code within a single dialog cannot reference more than one logical record that includes fields from a given subschema record.

16.3.2 WHERE clause

A WHERE clause is used to specify criteria for selection of one or more occurrences of a logical record that is the object of an ERASE, MODIFY, OBTAIN, or STORE command. A WHERE clause is also used to direct LRF to a particular logical record path.

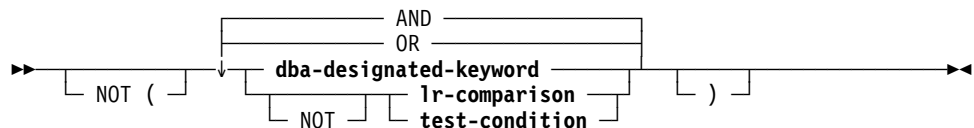
Considerations

- A WHERE clause is specified in the form of an expression that consists of one or more conditions to be tested.
 - Multiple conditions are combined with the logical operators AND and OR.
 - The logical operator NOT can precede a single condition or a compound condition that is enclosed in parentheses. NOT specifies the opposite of the condition.
- A test condition is expressed as a comparison or a keyword.
- A logical record occurrence is selected only if the entire expression evaluates as true.
- Operators in a conditional expression are evaluated one at a time, from left to right, in order of precedence.
 - The default order of precedence is the same as that described for other conditional expressions discussed in Chapter 8, “Conditional Expressions.”

16.3.3 Conditional expression

Purpose: The conditional expression of the WHERE clause is used when the process command syntax specifies lr-conditional-expression.

Syntax



Parameters

NOT

Specifies that the opposite of the condition fulfills the test requirements.

The opposite of the entire conditional expression can be specified by enclosing the expression in parentheses and preceding it with NOT.

dba-designated-keyword

Specifies a keyword, defined in the subschema by the database administrator (DBA), that directs LRF to a particular logical record path. The selected path must be associated with the object logical record.

lr-comparison

Specifies a comparison expression that establishes criteria used to select occurrences of the object logical record.

Syntax for the comparison expression is shown later in this chapter.

test-condition

Specifies a condition to be tested, such as command status or cursor position.

►► Test conditions are described in Chapter 8, “Conditional Expressions.”

AND

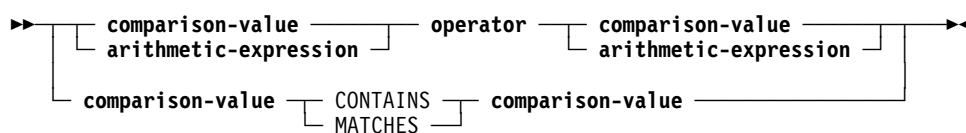
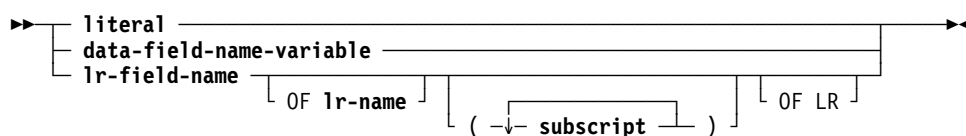
Specifies that the expression is true only if the outcome of both of the conditions being tested is true.

OR

Specifies that the expression is true if the outcome of either one or both of the conditions being tested is true.

16.3.4 Comparison expression

Purpose: Used to compare two values or to compare two character strings to determine if the first string matches or contains the second string.

Syntax*Expansion of comparison-value***Parameters****comparison-value**

Specifies the value to be compared.

Expanded syntax for comparison-value is shown above immediately following the comparison expression syntax.

arithmetic-expression

Specifies an arithmetic expression, according to the rules presented in Chapter 6, “Arithmetic Expressions.”

operator

The comparison operators are:

Operator	Synonym	Meaning
EQ	=	Equal
NE		Not equal to
GT	>	Greater than
LT	<	Less than
GE		Greater than or equal to
LE		Less than or equal to

CONTAINS

Searches the left operand for an occurrence of the right operand.

The length of the right operand must be less than or equal to the length of the left operand, and both operands must be EBCDIC or unsigned zoned decimal data types. If the right operand is not entirely contained in the left operand, the outcome of the comparison is false.

MATCHES

Compares the left operand to the right operand, one character at a time, beginning with the leftmost character in each operand. The right operand can contain mask characters, as follows:

- @ -- Matches any alphabetic character
- # -- Matches any numeric character
- * -- Matches any character

Any other character in the right operand matches only itself in the left operand.

literal

A user-supplied variable, expressed as a numeric constant, or the character string itself, enclosed in single quotation marks.

data-field-name-variable

Specifies the name of a variable data field, according to the rules presented in Chapter 11, “Variable Data Fields.”

lr-field-name

Specifies the name of a field in a Logical Record Facility record known to the subschema associated with the dialog.

OF lr-name

Specifies the name of the record that contains the field referenced by *lr-field-name*.

This clause is required only if the named field is not unique among the records known to the dialog.

subscript

Specifies the applicable occurrence of the field referenced by *lr-field-name*. This can be a variable field containing the applicable occurrence, the occurrence itself, or an expression.

This clause applies only if the named field is defined as a multiply-occurring field.

OF LR

Specifies that the value of the named field at the time that the request is issued is used throughout processing of the request.

If the value of the field changes during processing, LRF continues to use the original value. If OF LR is not specified and the value of the field changes during processing of the request, the new value in the dialog's record buffer is used for any further processing of the request.

Usage:

Considerations: Both the left and right operands must be EBCDIC or unsigned zoned decimal data types. The length of the string that is compared is set to the length of the shorter of the two operands. If a character in the left operand does not match the corresponding character in the right operand, the outcome of the comparison is false.

16.3.5 ERASE

Purpose: Deletes record occurrences.

The execution of an ERASE command does not necessarily result in the deletion of all or any of the database records used to create the object Logical Record Facility database-access record. The path selected to service the ERASE request performs only the database-access operations specified in the subschema.

Syntax

```
➤➤ ERASE lr-name [ WHERE lr-conditional-expression ]
      [ error-expression ] . ➤➤
```

Parameters**lr-name**

Specifies occurrences of the logical record used for database access.

Lr-name must be known to the dialog's subschema.

WHERE lr-conditional-expression

Specifies the selection criteria to be applied to the logical record request.

Syntax for lr-conditional-expression is described earlier in this section.

error-expression

Specifies the status codes that are returned to the dialog.

►► Error expressions are described in Chapter 10, “Error Handling.”

Example: The ERASE command in the following example deletes the occurrence of CUST-ORDER-LR with the specified customer and order numbers:

```
ERASE CUST-ORDER-LR
  WHERE CUST-NUMBER EQ '1234567890'
  AND ORD-NUMBER EQ '7654321'
  AND DELETE-ORDER.
```

The DBA-designated keyword, DELETE-ORDER, directs processing to a path that retrieves the applicable occurrence of the CUST-ORDER-LR logical record and deletes the specified order information from the database.

16.3.6 MODIFY

Purpose: Modifies field values in a record occurrence.

Syntax:

```
►►— MODIFY lr-name —————►
                        └ WHERE lr-conditional-expression ┘

► └ error-expression ┘ . —————►◄
```

Parameterslr-name

Specifies the name of the logical record.

Lr-name must be known to the dialog's subschema.

WHERE lr-conditional-expression

Specifies the selection criteria to be applied to the logical record request.

Syntax for lr-conditional-expression is described earlier in this section.

error-expression

Specifies the status codes that are returned to the dialog.

►► Error expressions are described in Chapter 10, “Error Handling.”

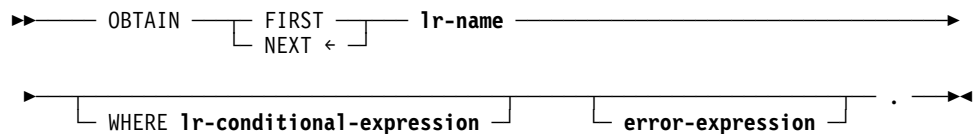
Example: The statements in the following example update an occurrence of the logical record CUST-ORDER-LR by specifying a new customer name and a new required date for the associated order:

```
OBTAIN FIRST CUST-ORDER-LR
  WHERE CUST-NUMBER EQ '1234567890'
    AND ORD-NUMBER EQ '7654321'.
MOVE NEW-CUST-NAME TO CUST-NAME.
MOVE NEW-DATE-REQ TO ORD-DATE-REQ.
MODIFY CUST-ORDER-LR.
```

16.3.7 OBTAIN

Purpose: Retrieves logical record occurrences.

Syntax:



Parameters

FIRST

Retrieves the first occurrence of the named logical record that meets the selection criteria specified in the WHERE clause.

NEXT

Retrieves the next occurrence of the named logical record that meets the selection criteria specified in the WHERE clause.

NEXT is the default when neither FIRST or NEXT is specified.

If the same selection criteria were not specified in a previous OBTAIN command, OBTAIN NEXT is equivalent to OBTAIN FIRST.

lr-name

Specifies the name of the logical record

Lr-name must be known to the dialog's subschema.

WHERE lr-conditional-expression

Specifies the selection criteria to be applied to the logical record request.

Syntax for lr-conditional-expression is described earlier in this section.

error-expression

Specifies the status codes that are returned to the dialog.

►► Error expressions is described in Chapter 10, "Error Handling."

Usage:

Definition: Data from object logical-record fields is transferred to the buffer established in the dialog's record buffers. The OBTAIN command can be issued iteratively to retrieve a series of record occurrences that meet the criteria specified in a WHERE clause.

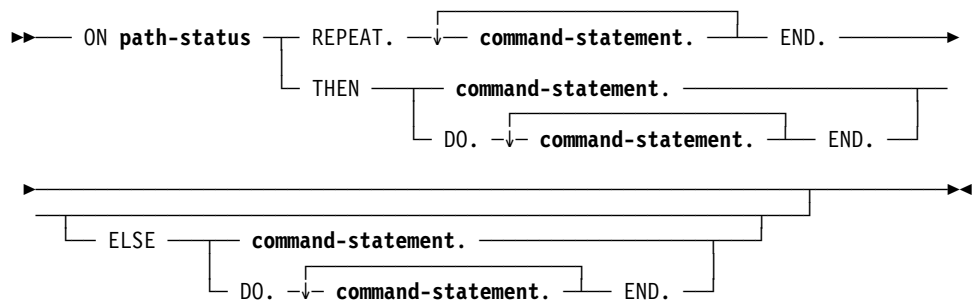
Example: The statements in the following example retrieve all occurrences of the logical record CUST-ORDER-LR for customer 1234567890:

```
OBTAIN FIRST CUST-ORDER-LR
  WHERE CUST-NUMBER EQ '1234567890'.
ON LR-NOT-FOUND THEN INVOKE 'CUSTCHEK'.
ON LR-FOUND
  REPEAT.
    OBTAIN NEXT CUST-ORDER-LR
      WHERE CUST-NUMBER EQ '1234567890'.
    .
    .
    .
  END.
DISPLAY.
```

16.3.8 ON command

Purpose: Indicates additional processing to be performed when a specified path status is returned by the Logical Record Facility following the execution of a LRF command.

Syntax:



Parameters

path-status

Tests whether Logical Record Facility returned the named path status. *Path-status* specifies a 1- to 16-character DBA-defined or standard path status defined for the path selected to service the previous logical record request.

REPEAT command-statement

Specifies the commands to be executed as long as LRF returns the named path status.

REPEAT begins a processing loop; END terminates the loop. Each command is executed sequentially before the path status is tested again.

Command-statement can be any valid CA-ADS process command, including another logical record command.

THEN command-statement

Specifies the commands to be executed if LRF returns the named path status.

Note: Multiple command statements must be preceded by DO and followed by END.

Command-statement can be any valid CA-ADS process command, including another logical command.

ELSE command-statement

Specifies the commands to be executed if LRF returns the named path status.

Command-statement can be any valid CA-ADS process command, including another logical record command.

Note: Multiple command statements must be preceded by DO and followed by END.

A given ON command statement can include only one ELSE clause, and that ELSE clause must match the most recent ON command not associated with an ELSE clause.

Usage

Path statuses: A path status, in the form of a 1- to 16-character unquoted string, indicates the result of a LRF request. LRF can return either a path status defined by the DBA in the subschema associated with the dialog or one of the standard path statuses. The standard path statuses are:

- **LR-FOUND** indicates that the logical record request was executed successfully. When LR-FOUND is returned, the dialog's error-status field contains 0000.
- **LR-NOT-FOUND** indicates that the object record cannot be found either because no such record exists or because all occurrences of the record have already been retrieved. When LR-NOT-FOUND is returned, the dialog's error-status field contains 0000.
- **LR-ERROR** indicates that a logical record request was issued incorrectly or that an error occurred in the processing of the path selected to service the request. When LR-ERROR is returned, the dialog's error-status field contains one of the status codes listed below.

Status code	Meaning
2001	The requested logical record was not found in the subschema (The path DML statement, EVALUATE, returns 0000 if true and 2001 if false)
2008	The object record is not in the dialog's subschema, or the specified request is not permitted for the named record
2010	The dialog's subschema prohibits access to logical records

Status code	Meaning
2040	The WHERE clause in an OBTAIN NEXT command directed LRF to a different processing path than did the WHERE clause in the preceding OBTAIN command for the same logical record
2041	The request's WHERE clause cannot be matched to a path in the dialog's subschema
2042	The logical record path for the request specifies return of the LR-ERROR status to the process
2043	Bad or inconsistent data was encountered in the logical record buffer during evaluation of the request's WHERE clause
2044	The request's WHERE clause does not include data required by the logical record path
2045	A subscript value in a WHERE clause is either less than zero or greater than its maximum allowed value
2046	One of the following conditions occurred during the evaluation of a WHERE clause: <ul style="list-style-type: none"> ■ Arithmetic overflow (fixed point, decimal, or exponent) ■ Arithmetic inflow (exponent) ■ Divide exception (fixed point, decimal, or floating point) ■ Significance exception
2063	The request's WHERE clause contains a keyword that exceeds the 16-character maximum
2072	The request's WHERE clause is too long to be evaluated in the available work area

Considerations

- One or more process commands can be specified to be executed once or iteratively, based on the returned path status. If an iterative sequence is used, the path status must change during processing to prevent uncontrolled looping.
- ON commands can be nested to any level and can be included in IF and WHILE command structures.
 - ▶▶ The IF and WHILE commands are described in Chapter 14, “Conditional Commands.”
- When coding ON commands, indentation should be used wherever possible to make the statement more readable and to ensure that the required clauses are properly matched.

Examples: The following examples test the path status before performing additional processing.

Example 1: Displaying messages when a record is not found

The statements in the following example display messages based on the path status returned after an attempt is made to retrieve a CUST-ORDER-LR logical record:

```
OBTAIN CUST-ORDER-LR
  WHERE CUST-NUMBER EQ '1234567890'.
ON NO-CUSTOMER
THEN
  DISPLAY MSG TEXT IS 'CUSTOMER NOT ON FILE'.
ON NO-ORDER
THEN
  DISPLAY MSG TEXT IS 'CUSTOMER HAS NO ORDERS'.
ON LR-NOT-FOUND
THEN
  DISPLAY MSG TEXT IS 'RECORD NOT FOUND'.
```

Example 2: Retrieving a record after a specified record

The statements in the following example retrieve VENDOR-LR logical records as long as the path status returned after the previous retrieval is VENDOR-CODE-010:

```
OBTAIN FIRST VENDOR-LR.
ON VENDOR-CODE-010
  REPEAT.
    OBTAIN NEXT VENDOR-LR.
    .
    .
    .
  END.
DISPLAY.
```

16.3.9 STORE

Purpose: Stores new occurrences of logical records.

Syntax:

```

►— STORE lr-name —┐
                    └─ WHERE lr-conditional-expression ┘
                    .
┐
└─ error-expression ┘

```

Parameters

lr-name

Specifies the name of the Logical Record Facility record.

Lr-name must be known to the dialog's subschema.

WHERE lr-conditional-expression

Specifies the selection criteria to be applied to the logical record request.

Syntax for lr-conditional-expression is described earlier in this section.

error-expression

Specifies the status codes that are returned to the dialog.

►► Error expressions are described in Chapter 10, "Error Handling."

Usage:

Considerations: The execution of a STORE command does not necessarily result in new occurrences of all or any of the database records used to create the object logical record. The path selected to service the STORE request performs only the database access operations specified in the subschema.

For example, CUST-ORDER-LR comprises fields from the CUSTOMER, PRODUCT, ORDOR, and ITEM records. A new CUST-ORDER-LR logical record is stored for each new customer order; however, only new occurrences of the ORDOR and ITEM records are actually added to the database. The CUSTOMER and PRODUCT records already exist in the database.

Example: The statements in the following example store a new occurrence of the logical record CUST-ORDER-LR for customer 1234567890. The DBA-designated keywords NEW-ORDER and NEW-ITEM direct LRF to the logical record paths that store new order and new item information, respectively.

```
MOVE ORDER-NEW TO ORDOR.  
STORE CUST-ORDER-LR  
  WHERE CUST-NUMBER EQ '1234567890' AND NEW-ORDER.  
  .  
  .  
  .  
  
MOVE ITEM-NEW TO ITEM.  
MODIFY CUST-ORDER-LR  
  WHERE NEW-ITEM.
```


Chapter 17. Map Commands

- 17.1 Overview 17-3
- 17.2 Map modification commands 17-4
- 17.3 Attributes Command 17-5
- 17.4 CLOSE 17-10
- 17.5 MODIFY MAP 17-12
- 17.6 Pageable maps 17-21
 - 17.6.1 Areas of a pageable map 17-21
 - 17.6.2 Map paging session 17-22
 - 17.6.3 Map paging dialog options 17-27
 - 17.6.4 GET DETAIL 17-28
 - 17.6.5 PUT DETAIL 17-30
 - 17.6.6 Creating or modifying a detail occurrence of a pageable map 17-32
 - 17.6.7 Specifying a numeric value associated with an occurrence 17-32
 - 17.6.8 Specifying a message to appear in the message field of an occurrence 17-32

17.1 Overview

Online maps (CA-ADS) and file maps (CA-ADS/Batch) are created and stored in the data dictionary using the CA-IDMS mapping facility. Map modification commands change the copy of the map maintained for a particular dialog, not the stored map definition.

►► Further information on maps and map attributes can be found in *CA-IDMS Mapping Facility*.

Map commands: CA-ADS map commands are used to adjust maps to meet the processing requirements of individual dialogs at run time. Pageable map commands are used to create, retrieve, and modify detail occurrences of a pageable map.

The map modification and pageable map commands are summarized in the following table. Each command is presented alphabetically later in this section.

Summary of map modification and pageable map commands

Type	Command	Description
Map modification commands	Attributes	Modifies the display intensity or the protected/unprotected specification of one or more map data fields, providing an alternative format to the MODIFY MAP command for these attributes
	CLOSE	Closes the dialog input and output file maps (batch only)
	MODIFY MAP	Modifies a map's write control character (WCC) options and specifies attributes of one or more map data fields
Pageable map detail commands	GET DETAIL	Retrieves a modified detail occurrence
	PUT DETAIL	Creates or modifies a detail occurrence

17.2 Map modification commands

Map modification commands are used to change a map to meet processing requirements of individual dialogs at run time. Single or multiple attributes can be changed globally or on a field-specific basis. Requested map modifications can be designated as temporary or permanent. Temporary changes apply only to the next display of the map. Permanent changes apply as long as the dialog remains operative in the application thread.

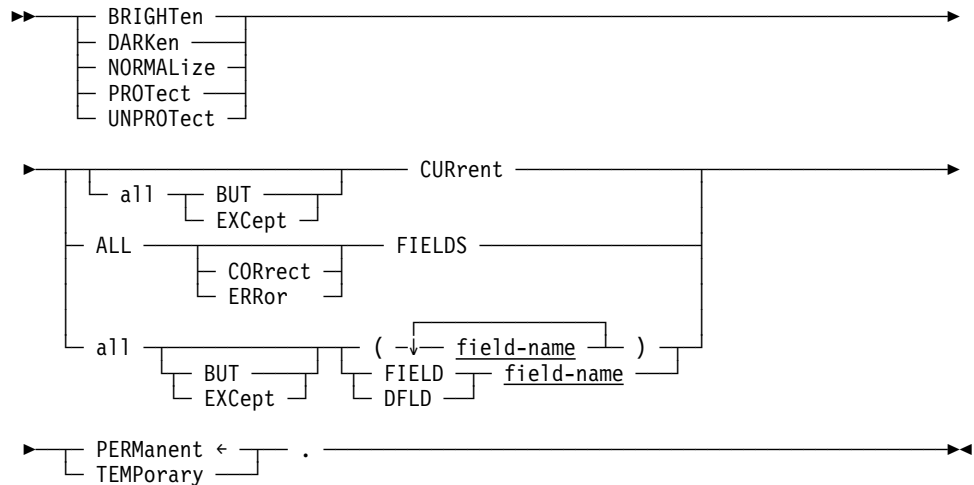
Pageable map considerations: For a pageable map, the following considerations apply:

- Permanent map modifications to detail area map fields modify only detail occurrences referenced by subsequent PUT DETAIL commands.
- Temporary map modifications to detail area map fields modify only the detail occurrence referenced by the next PUT DETAIL command. If temporary modifications are to apply in subsequent PUT DETAIL commands, the appropriate map modification commands must be repeated.
- Temporary map modifications to header and footer map data fields apply only to the first display of the map following the map modification. Temporary map modifications to fields in a detail occurrence apply only to the first display of that occurrence following map modification.

17.3 Attributes Command

Purpose: Modifies a map attribute for one or more map fields.

Syntax:



Parameters

BRIGHTen

Displays the specified map fields at brighter-than-normal intensity. A brightened field appears highlighted on the terminal screen.

DARKen

Displays the specified map fields at darker-than-normal intensity. Characters in a darkened field do not appear on the terminal screen.

NORMALize

Displays the specified map fields at normal intensity.

PROtect

Enables the input protect attribute for the specified map fields. The user cannot enter, modify, or delete data in the specified fields.

UNPROtect

Disables the input protect attribute for the specified map fields. The user can enter, modify, or delete data in the specified fields.

CURrent

Modifies the current map data field only. The current map data field is determined by the most recent map modification command or map field status condition test:

- **Map modification command** — The current field is the last field named in an explicit list of map fields or the last map field modified in an implicit list.

An implicit list results from specifying the FOR ALL BUT clause or the FOR ALL CORRECT/ERROR FIELDS clause in the map modification command.

An implicit list of map fields is ordered in the sequence in which the fields are defined in the map that is, top to bottom, left to right).

- **Map field status condition test** — The current field is the last field tested in the explicit or implicit list of fields. An implicit list results from specifying the ALL/ANY/NONE/SOME FIELDS clause in the map field status condition test.

►► For more information, see Chapter 8, “Conditional Expressions.”

The runtime system tests the fields in an explicit list from left to right, and tests the fields in an implicit list in the order in which the fields are defined in the map (that is, top to bottom, left to right).

Note that a status condition test ends at the first map field that determines the result of the test. For example, a test is run to determine if any fields in a list are truncated; the test stops at the first field that is truncated, and that field becomes the current map field.

all BUT

Modifies all map data fields except the current field.

EXCEPT can be used in place of BUT.

ALL FIELDS

Accompanies the fields to be modified when CORRECT or ERROR is specified.

CORrect

Modifies all map data fields set to be correct by the automatic error-handling facility or the dialog.

ERRor

Modifies all map data fields set to be correct by the automatic error-handling facility or the dialog.

If CORRECT or ERROR is not specified, all map data fields are modified.

all BUT

Introduces the fields to be modified.

The optional keyword BUT modifies all map data fields except the field or fields specified by *field-name*.

EXCEPT can be used in place of BUT.

FIELD field-name

Specifies the map data field to be modified.

DFLD can be used in place of FIELD.

PERManent

Specifies permanent modification.

The modification applies to each display of the map associated with the current dialog as long as the dialog remains operative in the application thread. In a pageable map, a modification to a map data field of a detail line occurrence applies throughout the map paging session.

PERMANENT is the default when neither TEMPORARY or PERMANENT is specified.

TEMPorary

Specifies temporary modification.

The modification applies only to the next display of the map associated with the current dialog. In a pageable map, a modification to a map data field of a detail line occurrence applies only to the next time the detail line occurrence is displayed on the screen during the map paging session.

Usage:

Definition: The attributes command provides an alternative format to the MODIFY MAP command for modification of display intensity or protected status of one or more map data fields. Only one attribute can be specified in a single attribute command.

The MODIFY MAP command can be used to modify multiple attributes. MODIFY MAP is discussed later in this section.

Example: The statements in the example below make up part of a response process that adds a new CUSTOMER record occurrence to the database. If the user enters a customer number that is already assigned, the screen is redisplayed with the CUST-NUMBER field in bright intensity:

```
FIND CALC CUSTOMER.  
IF DB-STATUS-OK  
THEN  
  DO.  
    BRIGHTEN FIELD CUST-NUMBER TEMPORARY.  
    DISPLAY MESSAGE TEXT IS  
      'CUSTOMER NUMBER ALREADY ASSIGNED.'  
  END.  
ELSE  
  STORE CUSTOMER.  
  DISPLAY MESSAGE TEXT IS  
    'CUSTOMER HAS BEEN ADDED.'
```

17.4 CLOSE

Purpose: (CA-ADS/Batch only) Closes the dialog input and output file maps.

Syntax:

```
➤ CLOSE [ BOTH | INPUT | OUTPUT ] file MAPs . ➤
```

Parameters

BOTH

Specifies the dialog's input and output file maps.

BOTH can be specified even if the dialog has only an input or an output file map.

BOTH is the default when no other option is specified.

INPUT

Specifies the dialog's input file map.

OUTPUT

Specifies the dialog's output file map.

Usage:*Considerations*

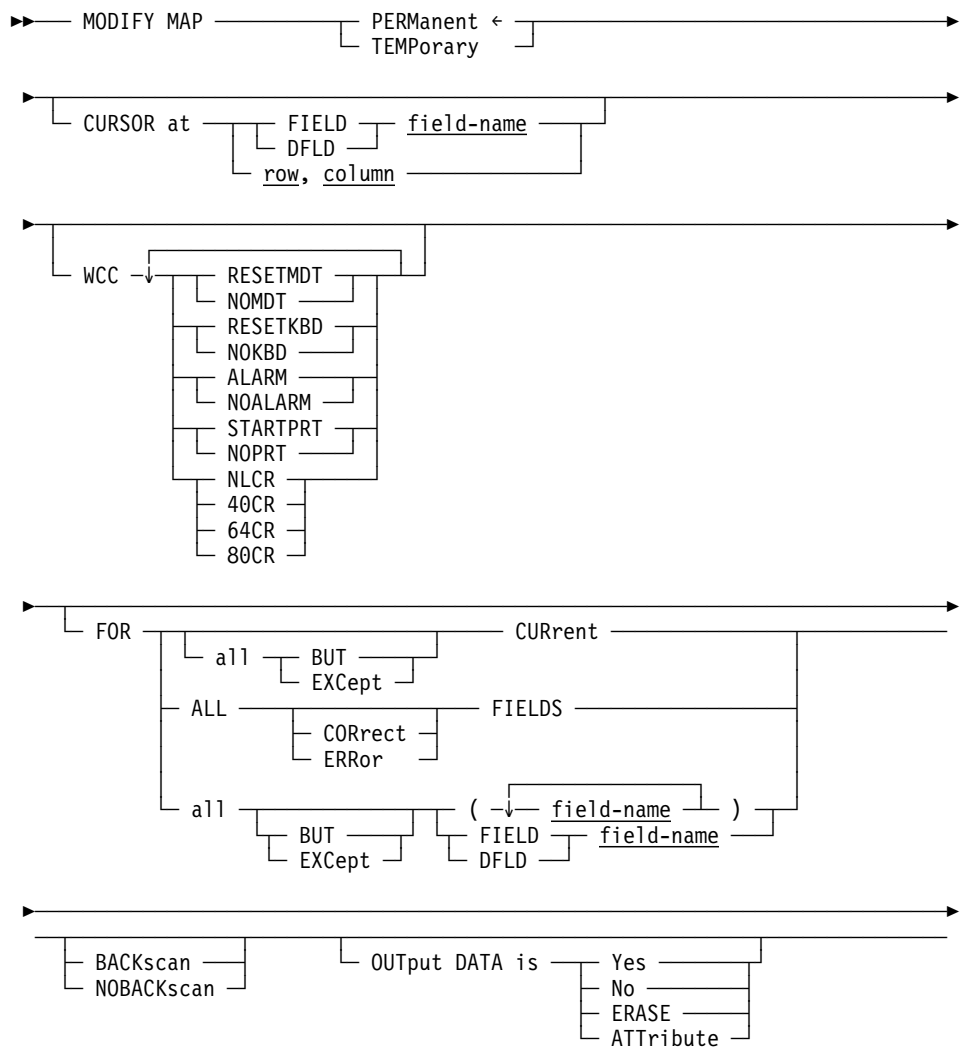
- The runtime system automatically closes the files if an application terminates with files still open:
 - A CLOSE command logically closes a file only if other dialogs using different maps have accessed the same file.
 - A CLOSE command must be issued for each map to physically close a file.
- The CLOSE command is required when closing a file before the application terminates, as in the following cases:
 - The application has been reading from or writing to a file and is required to start over at the beginning of the file.
 - An output file to which records were written is to be read as an input file.
 - A run-unit commit is performed by a COMMIT command or at the end of a run unit.

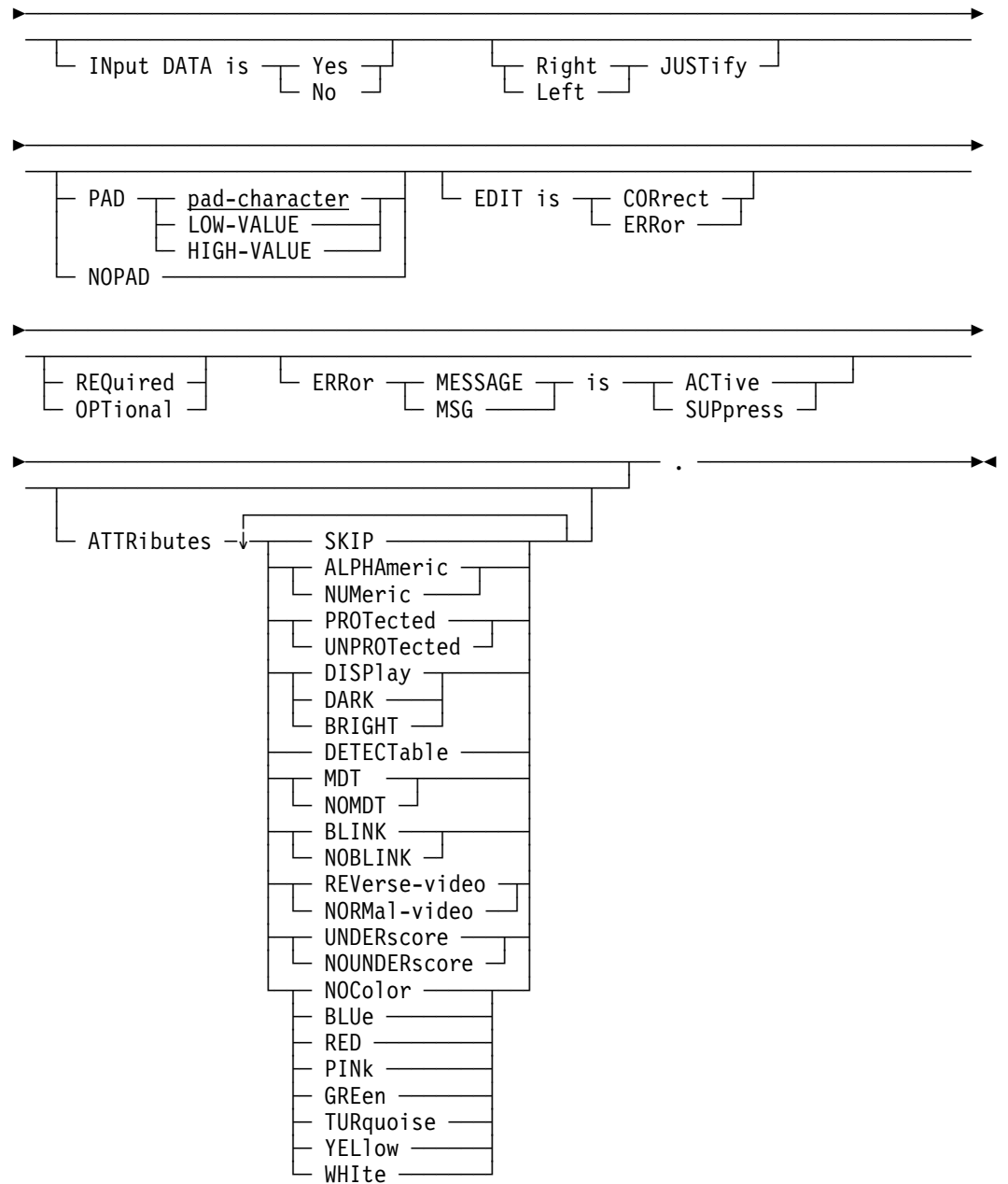
If a COMMIT command is issued, but not all files used in the application are closed, the runtime system either takes no action, sends a warning message to the log, or abends the application, as specified at system generation or at run time. The default action is abend.

17.5 MODIFY MAP

Purpose: Modifies a map write control character (WCC) options and specifies attributes for map data fields.

Syntax





Parameters

PERManent

Specifies permanent modification.

The modifications apply to each display of the map as long as the dialog remains operative in the application thread. In a pageable map, a modification to a map data field of a detail line occurrence applies throughout the map paging session.

PERMANENT is the default when neither TEMPORARY nor PERMANENT is specified.

TEMPorary

Specifies temporary modification.

The modifications apply only to the next display of the map. In a pageable map, a modification to a map data field of a detail line occurrence applies only to the next time the detail line occurrence is displayed on the screen during the map paging session.

CURSOR

Specifies the cursor position on the terminal screen when the map is displayed.

FIELD field-name

Positions the cursor at the beginning of the named map field.

Field-name specifies a data field in the map associated with the current dialog.

DFLD can be used in place of FIELD.

row

Either the name of a variable data field that contains the row number or the row number itself, expressed as a numeric constant.

column

Either the name of a variable data field that contains the column number or the column number itself, expressed as a numeric constant.

The specified row and column numbers must be 1- to 16-digit unsigned integers and must be valid for the terminal in use. The row and column specifications must be separated by a blank or a comma.

WCC

Modifies the write control character (WCC) specifications for the map associated with the current dialog.

RESETMDT

The modified data tags (MDTs) are turned on.

NOMDT

The modified data tags (MDTs) are not turned on.

An MDT marks a data field for transmission to the dialog whether or not it is modified by the user.

RESETKBD

The keyboard is unlocked when the map is displayed.

NOKBD

The keyboard remains locked when the map is displayed.

ALARM

If installed, the terminal's audible alarm will sound when the map is displayed.

NOALARM

Even if installed, the terminal's audible alarm will not sound when the map is displayed.

STARTPRT

The contents of the terminal buffer are printed when the map is displayed.

NOPRT

The contents of the terminal buffer are not printed when the map is displayed.

Note: This specification is meaningful only when a 3280-type printer is in use.

NLCR

No line formatting is performed on the printer output. The printer advances to a new line only when the new line (NL) and carriage return (CR) characters occur.

40CR

Printer output is formatted into 40 characters per line.

64CR

Printer output is formatted into 64 characters per line.

80CR

Printer output is formatted into 80 characters per line.

Note: This specification is meaningful only if the STARTPRT option above is specified.

If the MODIFY MAP command is used to alter any WCC option, *all* WCC options are overridden by the command. Unspecified WCC options default, as follows:

- RESETMDT/NOMDT defaults to NOMDT.
- RESETKBD/NOKBD defaults to NOKBD.
- ALARM/NOALARM defaults to NOALARM.
- STARTPRT/NOPRT defaults to NOPRT.
- NLCR/40CR/64CR/80CR has no default.

FOR

Specifies the map data fields being modified.

all BUT CURrent

Modifies all map data fields except the current field.

EXCEPT can be used in place of BUT.

ALL FIELDS

Introduces which map data fields are to be modified.

CORrect

Modifies all map data fields set to be correct by the automatic error-handling facility or the dialog.

ERRor

Modifies all map data fields set to be correct by the automatic error-handling facility or the dialog.

If CORRECT or ERROR is not specified, all map data fields are modified.

all BUT

Introduces the fields to be modified.

The optional keyword BUT modifies all map data fields except the field or fields specified by *field-name*.

EXCEPT can be used in place of BUT.

FIELD field-name

Specifies the map data field to be modified.

DFLD can be used in place of FIELD.

BACKscan

The contents of the designated map fields are displayed without trailing blanks. Characters remaining from the previous display of the map may appear in any unused positions.

NOBACKscan

The contents of the designated map fields are displayed with trailing blanks.

OUTput DATA is

Clause introducing selections which determine whether data from the dialog's record buffers and attribute specifications are transmitted to the designated map fields when the map is displayed. Attribute specifications include all attributes that can be specified in conjunction with the ATTRIBUTES keyword of the MODIFY MAP command.

Yes

Data and attribute specifications are transmitted.

No

Data is not transmitted. Data remaining from the previous display of the map appears in the designated map fields. Attribute specifications for a designated map field are transmitted only if one of the following conditions is met:

- The map being displayed is different than the map previously displayed.
- The designated map field is in error.

ERASE

Data is not transmitted, and data remaining from the previous display of the map is erased from the designated map fields. Attribute specifications are transmitted.

ATtribute

Attribute specifications are transmitted, but data is not. Data remaining from the previous display of the map appears in the designated map fields.

INput DATA is

Clause introducing selections which determine whether data entered in the specified map fields is transmitted to the dialog's record buffers.

Yes

Data in the designated map fields is transmitted to the dialog's record buffers.

No

Data in the designated map fields is not transmitted to the dialog's record buffers.

JUSTify

Introduces how data entered in the designated map fields is justified before it is transmitted to the dialog's record buffers.

Note: This specification is meaningful for nonnumeric fields only.

Right

Data in the designated map fields is right justified.

Left

Data in the designated map fields is left justified.

PAD pad-character

Specifies whether data entered in the designated map fields is padded before it is transmitted to the dialog's record buffers.

Data is padded on the left (if RIGHT JUSTIFY is specified) or on the right (if LEFT JUSTIFY is specified) with the specified pad character.

Pad-character is either the name of a variable data field that contains the pad character or the actual pad character, enclosed in single quotation marks.

NOPAD

Data in the designated map fields is not padded.

EDIT is

Specifies whether an error flag is set for the designated map fields.

ERRor

An error flag is set for the designated map fields.

CORrect

No error flag is set for the designated map fields.

Note: Error flags cannot be set permanently.

On a mapout operation, if any field is flagged to be in error, then for all fields both correct and incorrect) only attribute bytes are transmitted; no data is moved from program variable storage to the screen.

There is one exception to the above rule: on the initial display of a map by a CA-ADS dialog, all literals and data fields are transmitted even if a field is in error.

REQuired

The user must enter data in the designated map fields.

OPTional

The user can enter data in the designated map fields, as applicable.

ERRor MESSAGE is

Specifies display or suppression of an error message associated with a field.

ACTive

Enables display of an error message.

A message is usually enabled after ERROR MESSAGE SUPPRESS is specified within a MODIFY MAP PERMANENT specification.

SUPpress

Disables display of an error message associated with a field.

When the map is redisplayed because of errors, the message defined for the map field will not be displayed even if the field contains edit errors.

Note: Autoedit errors detected on map in for detail fields within a pageable map cannot be suppressed unless you turn off autoedit.

ATTRibutes

Applies 3270- and 3279-type terminal display attributes to the designated map fields.

SKIP

Causes repositioning of the cursor over the designated map fields to the next unprotected field.

SKIP automatically assigns the NUMERIC and PROTECTED attributes (see below) to the designated map fields.

ALPHAMeric

The user can enter any data type characters.

Note: ALPHAMERIC cannot be specified if SKIP (see above) is specified.

NUMeric

The user can enter only numeric data type characters.

PROTECTED

The designated map fields are input protected. The user cannot enter, modify, or delete data.

UNPROTECTED

The designated map fields are not input protected. The user can enter, modify, or delete data.

Note: UNPROTECTED cannot be specified if SKIP (see above) is specified.

DISPlay

The designated map fields are displayed at normal intensity.

DARK

The designated map fields are displayed at darker-than-normal intensity.

Characters in a darkened field do not appear on the terminal screen. DARK cannot be specified if DETECTABLE (see below) is specified.

BRIGHT

The designated map fields are displayed at brighter-than-normal intensity. A brightened field appears highlighted on the terminal screen.

DETECTable

Specifies that the designated map fields are detectable by selector light pen.

Note: DETECTABLE cannot be specified if DARK (see above) is specified.

MDT

Modified data tags (MDTs) are turned on for the designated map fields when the map is displayed.

NOMDT

Modified data tags (MDTs) are not turned on for the designated map fields when the map is displayed.

BLINK

(3279-type terminals only) The designated map fields are displayed with blinking characters.

Note: BLINK cannot be specified if either REVERSE-VIDEO or UNDERSCORE (see below) is specified.

NOBLINK

(3279-type terminals only) Blinking characters are suppressed for the designated map fields.

REVerse-video

(3279-type terminals only) The designated map fields are displayed with dark characters on a light background.

Note: REVERSE-VIDEO cannot be specified if either BLINK (see above) or UNDERSCORE (see below) is specified.

NORMal-video

(3279-type terminals only) The designated map fields are displayed with light characters on a dark background.

UNDERscore

(3279-type terminals only) The designated map fields are underscored.

Note: UNDERSCORE cannot be specified if either BLINK or REVERSE-VIDEO (see above) is specified.

NOUNDERscore

(3279-type terminals only) The designated map fields are not underscored.

NOColor

(3279-type terminals only) The designated map fields are displayed with the default color of the terminal.

BLUE/RED/PINK/GREen/TURquoise/YELLOW/WHITE

(3279-type terminals only) The designated map fields are displayed with one of the seven available color attributes.

Usage:*Considerations*

- Multiple attributes to be modified can be specified in a single MODIFY MAP command. All indicated modifications apply to all specified map data fields in the command.

If multiple attributes are specified, they must be separated by commas or blanks.

- The following rules apply to attributes and WCC options that are omitted from a MODIFY MAP command:
 - If an attribute that is not a WCC option is omitted, the attribute remains as defined at map compilation time or as set by a previous modification designated as PERMANENT.
 - If any WCC option is altered by the MODIFY MAP command, *all* WCC options are overridden by the command. Unspecified WCC options are assigned the default values listed in the syntax rules below.
- The ERROR MESSAGE clause of the MODIFY MAP statement allows suppression of a default error message and display of a more appropriate message. For example, the following error message can be displayed for a part-number field in an order entry application:

THE SPECIFIED PART CANNOT BE MAILED

Note: Pageable maps cannot have the error message suppressed on map in.

Example: The following statements are part of a response process that adds a new CUSTOMER record occurrence to the database. The CUST-NUMBER field is required when adding a customer. If the user does not enter a customer number, an error flag is set for the CUST-NUMBER field and the field is made required:

```
IF CUST-NUMBER EQ SPACES
THEN
  DO.
    MODIFY MAP TEMPORARY FOR FIELD CUST-NUMBER EDIT ERROR.
    MODIFY MAP PERMANENT FOR FIELD CUST-NUMBER REQUIRED.
    DISPLAY MESSAGE TEXT IS
      'CUSTOMER NUMBER REQUIRED WHEN ADDING CUSTOMER.'.
  END.
ELSE
  MODIFY MAP TEMPORARY FOR FIELD CUST-NUMBER EDIT CORRECT OPTIONAL.
```

17.6 Pageable maps

A pageable map is a map that contains multiple occurrences of a set of map fields. Each occurrence of the multiply-occurring set is called a **detail occurrence**.

A pageable map can contain more detail occurrences than can fit on the user's screen at one time. The runtime system stores detail occurrences in the order in which they are created by pageable map commands, and divides them into pages, based on the number of occurrences that can fit on the screen. One page of occurrences can be displayed on the screen at a time.

An example of a pageable map is one that displays information about a department and lists all the employees within the department. The set of map fields related to employee information occurs once for each employee to be listed. These detail occurrences of employee information are created at run time by pageable map commands and can be displayed to the user one page at a time.

17.6.1 Areas of a pageable map

A pageable map is divided into three areas.

Header area: The **header area** (optional) is located across the top of the screen and contains one or more rows of map fields associated with header information. The header area information is displayed whenever the map is displayed.

Detail area: The **detail area** (required) is located across the middle of the screen and contains the detail occurrences. Detail occurrence map fields are defined in the detail area only once. At run time, the number of detail occurrences that are displayed in the detail area depends on the space available on the screen after accounting for the header and footer information.

Footer area: The **footer area** (optional) is located across the bottom of the screen and contains one or more rows of map fields associated with footer information. The footer information is displayed whenever the map is displayed.

For example, a pageable map used to display a department record and all associated employee records might contain the following information:

- **Header area** — The title of the map and department information
- **Footer area** — A message field, the map page, and information about how to page through the map
- **Detail area** — Detail occurrences of employee information

DEPT. ID: _____

EMP. ID: _____ LAST NAME: _____ ACTION CODE: _____
START DATE: _____ MESSAGE: _____

PAGE: _____

17.6.2 Map paging session

A **map page** refers to the header and footer map fields and to a page of detail occurrences.

When a pageable map is displayed, the page of occurrences that appears in the detail area is determined by the current value of the \$PAGE system-supplied data field. For example, given a screen that can hold ten occurrences, if \$PAGE equals 1, occurrences 1 through 10 are displayed; if \$PAGE equals 2, occurrences 11 through 20 are displayed; and so forth. Actions taken by the user and commands issued by premap and response processes can modify the value of \$PAGE.

►► For more information, see Chapter 11, “Variable Data Fields.”

Beginning a map paging session: A **map paging session** begins when a dialog associated with a pageable map begins execution. A map paging session ends when the application terminates or when a dialog passes control to another dialog under any of the following conditions:

- The dialog receiving control is associated with a different pageable map than the dialog that initiated the map paging session
- The dialog receiving control has different map paging dialog options than the dialog that initiated the map paging session
- The dialog that initiated the map paging session issues a TRANSFER command
- The dialog receiving control is at a level higher than the dialog that initiated the map paging session

Note: The first two conditions do not apply when the receiving dialog is not associated with a pageable map. In such cases, the map paging session continues, provided that the third or fourth condition is not met.

If none of the above conditions is met, the map paging session continues. Detail occurrences created during the session can be added to, displayed, and modified by dialogs associated with the pageable map. If the map paging session terminates, the runtime system deletes all detail occurrences created during the session.

One or more dialogs can be associated with the same pageable map in a given map paging session. During a map paging session, premap and response process commands can create, display, retrieve, and modify detail occurrences.

Considerations: The following considerations apply:

- **Detail occurrences are created** by PUT NEW DETAIL process commands. Detail occurrences are built from the values stored in the variable data fields to which the detail occurrence fields map.

The runtime system stores detail occurrences in the order in which they are created and divides them into pages, based on the number of detail occurrences that can fit on the screen at one time. A detail occurrence is displayed on the screen only when the map page to which the occurrence belongs is displayed.
- **A dialog process displays a map page to the terminal** as a result of either of the following actions:

- **A PUT NEW DETAIL command is issued** that creates the first detail occurrence of the second map page. The runtime system automatically displays the first map page, allowing the user to enter information.

The process that issues the PUT NEW DETAIL command continues to execute and can create additional detail occurrences. The process must issue a DISPLAY command to terminate processing. The runtime system does not process information entered during a pseudo-converse until the DISPLAY command is issued.

Note: In this case, the DISPLAY command does not send information to the terminal. Header and footer variable data fields should be primed before the first map page is displayed. If the map contains a message field in the header or footer area, any text for the message field should be specified once by issuing a PUT NEW DETAIL command before the first map page is displayed.

- **A DISPLAY command is issued**, except when the map has already been displayed as a result of a PUT NEW DETAIL command. The map page displayed is determined by the current value of \$PAGE.

►► For more information, see Chapter 11, “Variable Data Fields.”

- **The user can modify map data fields on the screen**, including header and footer data fields and detail occurrence fields of the current map page. Restrictions that apply include those specified in the map definition (such as the PROTECT specification), in the dialog definition (that is, the paging mode dialog option, UPDATE/BROWSE), and by process commands (such as the MODIFY MAP command).
- **The user can make a paging request** to specify the next map to be displayed by performing one of the following actions:
 - Pressing the control key associated with paging forward one page. The system generation default paging-forward key is PF8.
 - Pressing the control key associated with paging backward one page. The system generation default paging-backward key is PF7.
 - Changing the \$PAGE map field (if one is defined for the map) and pressing a control key other than the paging-forward key, paging-backward key, [Clear], [PA1], [PA2], or [PA3]
- **The user presses a control key**, including the paging-forward or paging-backward key, and the runtime system performs the following processing:
 - **Updates map data fields** — The runtime system updates its internal representation of the header and footer map data fields and updates detail occurrence fields to reflect changes made by the user. No updates are performed if [Clear], [PA1], [PA2], or [PA3] are pressed; these control keys do not transmit data.

Map field attributes set temporarily by the user or by map modification commands are reset. Attributes set permanently in the map definition or by map modification commands remain set.
 - **Updates \$PAGE** — If a paging request was made, the runtime system updates \$PAGE as follows:
 - Adds 1 to \$PAGE if the paging forward key was pressed and the current map page is not the last map page.
 - Subtracts 1 from \$PAGE if the paging backward key was pressed and the current map page is not the first map page.

- Moves the value entered in the \$PAGE map field to \$PAGE if the \$PAGE map field was changed and the control key pressed was not the paging forward key, the paging backward key, [Clear], [PA1], [PA2], or [PA3]. If the value entered in the \$PAGE field is less than the first map page or greater than the last map page, \$PAGE is set to the first or last page number.

\$PAGE determines the next map page to be displayed.

- **Determines the flow of control** — In a session that is not a map-paging session, the runtime system always attempts to initiate a function or response process when the user presses a control key. In a map paging session, the runtime system either attempts to initiate a function or response process, or instead displays the same or another map page. The action taken by the runtime system depends on the paging-type dialog option (NOWAIT/WAIT/RETURN), on whether a paging request was made, and whether any map field's modified data tag was set, as shown in the following table.

Flow of control in a map paging session:

Paging type	Paging request ¹		Nonpaging request	
	No MDT set	Any MDT set ²	No MDT set	Any MDT set ²
NOWAIT	Displays the requested map page	Displays the requested map page	Initiates a function or response process ³	Redisplays the same map page
WAIT	Displays the requested map page	Initiates a function or response process ³	Initiates a function or response process ³	Initiates a function or response process ³
RETURN	Initiates a function or response process ³	Initiates a function or response process ³	Initiates a function or response process ³	Initiates a function or response process ³

Notes:

¹ A paging request occurs when the user presses a control key associated with paging forward or backward or modifies the \$PAGE field, if one is defined for the map. If [Clear], [PA1], [PA2], or [PA3] is pressed, any modification to \$PAGE is ignored and is not considered as a paging request. If a paging request is not made, refer to the Nonpaging request columns.

² If the control key pressed is [Clear], [PA1], [PA2], or [PA3], refer to the No MDT set column under the applicable Paging/Nonpaging request column.

³ The function or response is selected as described under "Runtime flow of control" in Chapter 16, "Database Access Commands."

If the same or another map page is displayed, the user can modify map fields, make a paging request, and press a control key, as described above.

If a function or response process is initiated, the internal representations of the header and footer fields are mapped into their associated variable data fields.

- **Detail occurrences are retrieved**

by GET DETAIL process commands. A GET DETAIL command locates the occurrence to be retrieved, then moves the occurrence's fields into the variable data fields to which the fields map.

- **Only modified detail occurrences can be retrieved.** A detail occurrence is considered to be modified if it has the following two characteristics:

- Contains one or more map fields whose modified data tags (MDTs) are set at the time of the most recent pseudo-converse.
- Has yet to be retrieved since the most recent pseudo-converse. Once a modified detail occurrence has been retrieved, it is no longer considered to be modified.

Note that if a modified detail occurrence is not retrieved following a pseudo-converse, it is not automatically considered to be modified following a subsequent pseudo-converse. The detail occurrence must once again have the two characteristics listed above.

A detail occurrence that is not a modified detail occurrence cannot be retrieved by dialog process code.

- **Detail occurrence fields are modified in dialog processes** by PUT CURRENT DETAIL commands. A PUT CURRENT DETAIL command modifies the detail occurrence referenced by the most recent GET DETAIL or PUT DETAIL command.

- **Additional detail occurrences can be created** by PUT NEW DETAIL commands. New occurrences are stored at the end of the set of detail occurrences.

- **Detail occurrences cannot be deleted by process commands.** Detail occurrences are deleted as follows:

- If the backpage dialog option (described below) is NO, detail occurrences of previous map pages are deleted when a new map page is displayed.
- At the end of a map paging session, all detail occurrences are deleted.

17.6.3 Map paging dialog options

Map paging dialog options define parameters for a map paging session. Specification of options for a dialog are made during dialog definition. The map paging dialog options NOWAIT, BACKPAGE NO, and UPDATE cannot be specified together.

The following table lists available map paging dialog options.

Map paging dialog options

Option	Parameter	Description
Paging type	NOWAIT WAIT RETURN	Specifies the runtime flow of control when the user presses a control key, as described in the previous table.
Backpage	BACKPAGE YES	Allows the user to display a previous map page. The runtime system maintains the resources that describe the detail occurrences of previous pages.
	BACKPAGE NO	Prohibits the user from displaying a previous map page. The runtime system deletes all previous pages of detail occurrences when a new map page is displayed. The lowest page number is the first page that has not been deleted. ¹
Paging mode	UPDATE	Specifies that the terminal operator can modify map data fields, subject to restrictions specified in the mapping facility and by map modification process commands.
	BROWSE	Specifies that the user can modify only the \$PAGE and \$RESPONSE fields of the map. Map fields can still have their MDTs set in the map definition or by map modification commands.

Notes:

¹ On mapin from the terminal when backpageing is not allowed, if \$PAGE has been set to a value greater than the current map page, the runtime system flags all map pages

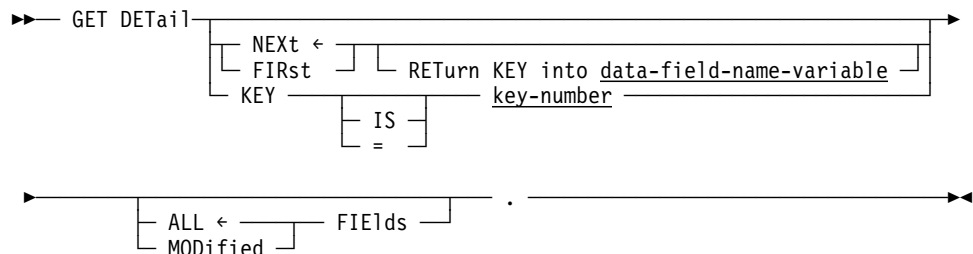
below \$PAGE for deletion. When the map is displayed again, these flagged pages are deleted, even if \$PAGE has been modified to a lower value in the interim.

17.6.4 GET DETAIL

Purpose: Retrieves a modified detail occurrence of a pageable map.

A GET DETAIL command can retrieve all the fields of a modified detail occurrence or only those fields whose MDTs are turned on.

Syntax:



Parameters

NEXt

Retrieves the first modified detail occurrence that follows the detail occurrence referenced by the preceding pageable map command.

The preceding pageable map command must follow the most recent pseudo-converse.

FIRst

Retrieves the first modified detail occurrence of the pageable map.

Note that the GET DETAIL FIRST command can be used repeatedly to retrieve all the modified detail occurrences of a pageable map. The first GET DETAIL FIRST command retrieves the first modified detail occurrence. Once retrieved, the occurrence is no longer considered as modified, and the second modified detail occurrence becomes the first modified detail occurrence. This modified detail occurrence can be retrieved by a subsequent GET DETAIL FIRST command, and so forth.

An end-of-data condition results if no pageable map command precedes the GET DETAIL command, if the preceding pageable map command resulted in an end-of-data or detail-not-found (see the KEY IS parameter below) condition or if the GET DETAIL command cannot find a modified detail occurrence before reaching the end of the set of detail occurrences.

RETurn KEY into data-field-name-variable

Specifies the numeric variable field into which the runtime system moves the binary fullword value (if any) associated with the detail occurrence being retrieved. A value is associated with a detail occurrence by specifying the KEY IS parameter in a PUT DETAIL command.

If no value is associated with the detail occurrence, *data-field-name-variable* is set to zero. *Data-field-name-variable* does not have to be a binary fullword.

KEY is key-number

Specifies the modified detail occurrence to be retrieved based on the numeric key value associated with the detail occurrence. A key value is associated with a detail occurrence by specifying the KEY IS parameter in a PUT DETAIL command.

Key-number is either the numeric variable data field or the numeric literal itself.

The runtime system finds the first detail occurrence associated with the key value specified by *key-number*. If the detail occurrence is a modified detail occurrence, it is retrieved. If the occurrence is not a modified detail occurrence, or if no detail occurrence with the specified key value is found, a detail-not-found condition is set.

ALL

Specifies all the fields of the modified detail occurrence to be retrieved.

ALL is the default when neither ALL or MODIFIED is specified.

MODified

Specifies only those fields whose MDTs are turned on to be retrieved.

If MODIFIED is specified, variable data fields that map to nonretrieved fields retain their previous values.

Usage*Considerations:*

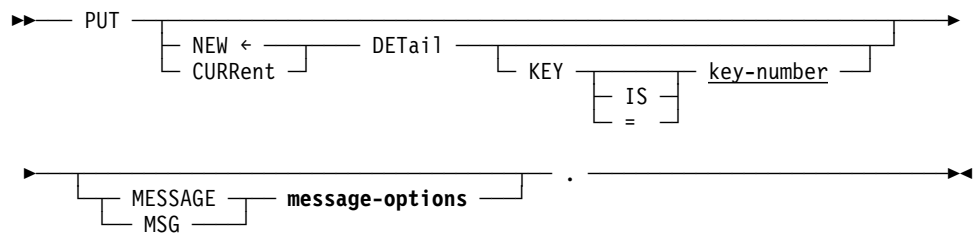
- The GET DETAIL command causes the runtime system to move the following:
 - The retrieved fields into the variable data fields to which they map
 - The page number of the retrieved detail occurrence into the \$PAGE system-supplied data field
 - The numeric key value (if any) associated with the occurrence into a specified field (optional)
- A GET DETAIL command can retrieve only a modified detail occurrence. A detail occurrence is considered modified if it has the following characteristics:
 - Contains one or more map fields whose modified data tags (MDTs) are turned on at the time of the most recent pseudo-converse.
 - Has yet to be retrieved. Once a modified detail occurrence has been retrieved, it is no longer considered modified.
- A detail occurrence that is not a modified detail occurrence cannot be retrieved by dialog process code.

17.6.5 PUT DETAIL

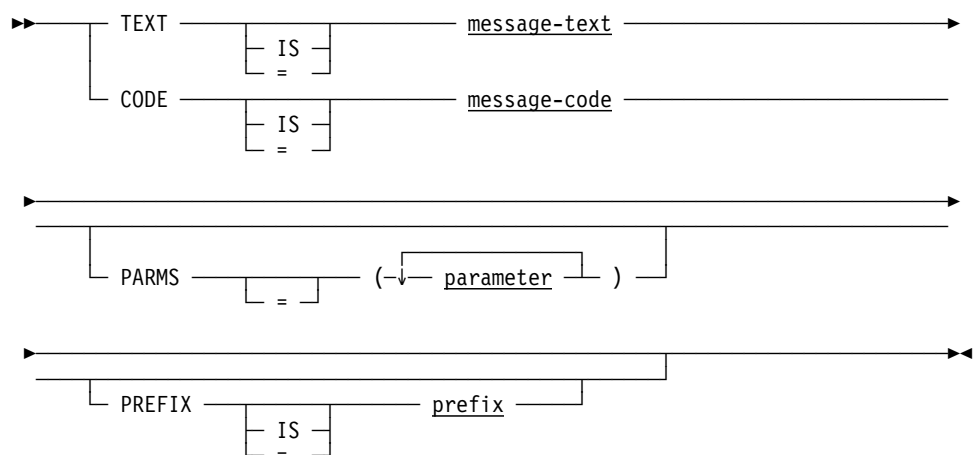
Purpose: The PUT DETAIL command:

- creates or modifies a detail occurrence of a pageable map
- specifies a numeric value to be associated with the occurrence
- specifies a message to appear in the message field of the occurrence.

Syntax:



Expansion of message-options



Parameters

NEW DETail

Creates a detail occurrence which is stored at the end of the set of detail occurrences.

NEW is the default when neither NEW or CURRENT is specified.

CURRENT DETail

Modifies the detail occurrence referenced by the most recent pageable map command.

After a pseudo-converse, a pageable map command must be issued to establish currency on a detail occurrence before a PUT CURRENT DETAIL command can be issued. If currency is not established, CA-ADS abnormally terminates the dialog.

KEY is key-number

Specifies the numeric value to be associated with the detail occurrence being created or modified.

Key-number is either the name of a variable data field or the number itself, expressed as a numeric constant.

Key-number replaces the numeric value (if any) previously associated with the detail occurrence. The numeric value is not displayed at the terminal, but is stored along with the detail occurrence as a binary fullword.

The KEY parameter can be used to store the database key of a subschema record associated with a detail occurrence. A GET DETAIL command can later retrieve the database key when it retrieves the detail occurrence, facilitating the retrieval of the subschema record.

MESSAge

Introduces the text or code of a message.

MSG can be used in place of MESSAGE.

message-options

Identifies message to be displayed.

Expanded syntax for message-options is shown above immediately following the PUT DETAIL syntax.

TEXT is message-text

Specifies the text of a message to be displayed in an online map's message field or sent to a batch application and a system log file.

Message-text specifies either the name of a variable data field containing the message text or the text string itself, enclosed in single quotation marks.

The text string can contain up to 240 displayable characters.

CODE is message-code

Specifies the message dictionary code of a message to be displayed in an online map's message field or sent to the log file in a batch application.

In a batch application, the message is also sent to the operator, if directed by the destination specified in the dictionary.

Message-code specifies either the name of a variable data field that contains the message code or the 6-digit code itself, expressed as a numeric literal.

PARMS = parameter

Specifies a replacement parameter for each variable field in the stored message identified by *message-code*.

Up to nine replacement parameters can be specified for a message. Multiple parameters must be specified in the order in which they are numbered and separated by blanks or commas.

Parameter specifies either the name of an EBCDIC or unsigned zoned decimal variable data field that contains the parameter value or the parameter value itself, enclosed in single quotation marks.

The parameter value must contain displayable characters. At run time, each variable data field in a stored message expands or contracts to accommodate the size of its replacement parameter. A replacement parameter can be a maximum of 240 bytes.

PREFIX is prefix

Overrides the default prefix of a dialog and a map.

Prefix specifies an EBCDIC or unsigned zoned decimal variable data field that contains a 2-character prefix or the 2-character prefix itself, enclosed in single quotation marks.

17.6.6 Creating or modifying a detail occurrence of a pageable map

After a PUT DETAIL command is executed, the map fields of a created or modified occurrence contain the values of the variable data fields to which they map. The created or modified occurrence appears on the user's screen when the map page to which it belongs is displayed.

Storage: The amount of storage available at run time to hold detail occurrences is specified at system generation with the PAGING STORAGE clause of the OLM statement. By default, the available storage is 10K bytes. If a PUT DETAIL command would cause storage overflow, the detail occurrence is not created and the \$MAXIMUM-DETAILS-PUT map paging condition is set. The \$MAXIMUM-DETAILS-PUT condition can be tested.

►► For more information about \$MAXIMUM-DETAILS-PUT, see Chapter 8, “Conditional Expressions.”

For information on calculating the storage required by a pageable map, refer to *CA-IDMS Mapping Facility*.

17.6.7 Specifying a numeric value associated with an occurrence

A numeric value, such as a database key, can be associated with a created or modified detail occurrence.

This value is not displayed to the user, but can be retrieved by a GET DETAIL command.

17.6.8 Specifying a message to appear in the message field of an occurrence

The text of a message or a code associated with a message that has already been defined in the message dictionary can be specified in a PUT DETAIL command. When the dialog is executed, the runtime system moves the appropriate message to the message field in the dialog's map.

A message field is defined by the \$MESSAGE map field.

►► For more information, refer to *CA-IDMS Mapping Facility*.

If no message field is defined for the detail area of the pageable map, the runtime system places the message in the header or footer message field or, if neither the header nor the footer has a message field, the runtime system ignores the message. If more than one message is placed in the header or footer message field, the messages are concatenated up to the length of the message field.

Considerations: The following considerations apply to specifying a message code in a PUT DETAIL command:

- Each system-supplied message in the data dictionary message area (DDLDCMSG) is identified by a six-digit code prefixed by the letters DC. For example, a request for message 987654 retrieves message DC987654.

User-defined messages added to the message dictionary can have a prefix other than DC and digits in the range 900001 through 999999.

- Each message in the message dictionary can be assigned a severity code. The severity code specifies the action CA-ADS takes when a message is retrieved. The following table lists the severity codes and their associated actions.

Severity code	Action
0	Processes the PUT DETAIL command
1	Snaps all CA-ADS resources and processes the PUT DETAIL command
2	Snaps all system areas and processes the PUT DETAIL command
3	Snaps all CA-ADS resources and terminates CA-ADS with a task abend code of D002
4	Snaps all system areas and terminates CA-ADS with a task abend code of D002
5	Terminates CA-ADS with a task abend code of D002
8	Snaps all system areas and terminates the DC system with an operating system abend code of 3996
9	Terminates the DC/UCF system with an operating system abend code of 3996

- A message in the message dictionary can contain one or more variable fields that are replaced with application-specific values at run time. In a PUT DETAIL command, the application developer can use the PARMS parameter to code replacement parameters for each variable field in a specified message.

Within the message definition in the dictionary, symbolic parameters are identified by an ampersand (&) followed by a two-digit numeric identifier. These identifiers can appear in any order. The position of the replacement values in the PARMS parameter must correspond directly to the two-digit numeric identifiers in the message; the first value corresponds to &01, the second to &02, and so forth. For example, assume that the stored message text is as follows:

```
THIS IS TEXT &01 AND &03 OR &02
```

The PARMS parameter reads PARMS=('A','B','C'). The resulting text would read as follows:

```
THIS IS TEXT A AND C OR B
```

- If the message is defined in the dictionary with more than one text line, only the first line appears in the map's message field.

Example: The example below illustrates the map and the premap and response processes of a dialog that:

- Lists the employees in a department one page at a time
- Allows the user to modify employee information and delete employees
- Updates the database based on the user's entries
- Redisplays the map with appropriate messages and allows the user to make further modifications

The **paging type** in this example is WAIT. If the user makes a paging request and no MDTs are set, the runtime system displays the requested page. If the user makes a nonpaging request or if any MDTs are set, the runtime system initiates the response process. The response process is associated with the control keys ENTER, FWD (paging forward), and BWD (paging backward).

The **pageable map** associated with the dialog is shown in the screen that follows. The following considerations apply to the detail area map fields:

- The fields are defined once. At run time, the number of occurrences of these fields that are displayed to the user at any one time depends on the number of occurrences that can fit on the screen between the header and footer areas.
- At run time, the fields map to variable data fields through detail occurrences:
 - PUT NEW DETAIL commands create detail occurrences from associated variable data fields.
 - When a map page is displayed, the detail occurrences for the page are displayed.
 - When the user presses a control key, the appropriate detail occurrence fields are updated.
 - GET DETAIL commands can retrieve modified detail occurrences into associated variable data fields.

1. Maps to DEPT-ID of DEPARTMENT database record
2. Maps to WK-EMP-ID through detail occurrence
3. Maps to WK-EMP-LNAME through detail occurrence
4. Maps to WK-ACTION through detail occurrence
5. Maps to WK-EMP-START-DATE through detail occurrence
6. Maps to \$MESSAGE through detail occurrence
7. Maps to \$PAGE system-supplied data field
8. Maps to WK-MESSAGE

- Obtains a DEPARTMENT record based on a CALC key passed from another dialog or function
- Obtains all associated EMPLOYEE records
- Creates a detail occurrence for each retrieved record
- Displays the first map page at the terminal

```
OBTAIN CALC DEPARTMENT.
IF DB-REC-NOT-FOUND
  THEN
    DO.
      MOVE 'DEPARTMENT NOT FOUND' TO WK-MESSAGE.
      DISPLAY.
    END.
MOVE SPACES TO WK-ACTION.
OBTAIN FIRST EMPLOYEE WITHIN DEPT-EMPLOYEE.
WHILE NOT DB-END-OF-SET
  REPEAT.
    MOVE EMP-ID TO WK-EMP-ID.
    MOVE EMP-LNAME TO WK-EMP-LNAME.
    MOVE EMP-START-DATE TO WK-EMP-START-DATE.
    ACCEPT DB-KEY INTO WK-KEY FROM CURRENCY.
    PUT NEW DETAIL KEY WK-KEY.

    IF $PAGE-READY
      THEN
        DO.
          MOVE 'MORE EMPLOYEES EXIST FOR THIS DEPT' TO WK-MESSAGE.
          DISPLAY.
        END.

    OBTAIN NEXT EMPLOYEE WITHIN DEPT-EMPLOYEE.
  END.

MOVE 'ALL EMPLOYEES DISPLAYED FOR THIS DEPT' TO WK-MESSAGE.
DISPLAY.
```

Sample response process: The **response process** shown below performs the following:

- Retrieves each modified detail occurrence.
- Updates the EMPLOYEE database accordingly.
- Modifies each retrieved detail occurrence:
 - Moves a confirming message to the message field
 - Initializes the action code
 - Protects the fields if the associated database record is deleted
- Redisplays the map. The value of \$PAGE is saved at the beginning of the response process and is restored at the end in order to display the page requested by the user. During the response process, \$PAGE is modified by GET DETAIL commands.

```
READY USAGE-MODE UPDATE.
MOVE $PAGE TO WK-PAGE.
GET DETAIL FIRST RETURN KEY WK-KEY.
WHILE NOT $END-OF-DATA
  REPEAT.
    OBTAIN EMPLOYEE DB-KEY IS WK-KEY.
    IF WK-ACTION EQ 'DEL'
      THEN
        DO.
          ERASE EMPLOYEE.
          PROTECT (WK-EMP-ID WK-EMP-LNAME
                    WK-EMP-START-DATE WK-ACTION) PERMANENT.
          MOVE SPACES TO WK-ACTION.
          PUT CURRENT DETAIL TEXT 'DELETED'.
        END.
      ELSE
        DO.
          MOVE WK-EMP-LNAME TO EMP-LNAME.
          MOVE WK-EMP-START-DATE TO EMP-START-DATE.
          MODIFY EMPLOYEE.
          MOVE SPACES TO WK-ACTION.
          PUT CURRENT DETAIL TEXT 'MODIFIED'.
        END.
    GET DETAIL NEXT RETURN KEY WK-KEY.
  END.
MOVE WK-PAGE TO $PAGE.
DISPLAY.
```


Chapter 18. Queue and Scratch Management Commands

- 18.1 Overview 18-3
- 18.2 Queue records 18-5
- 18.3 DELETE QUEUE 18-7
- 18.4 GET QUEUE 18-9
- 18.5 PUT QUEUE 18-12
- 18.6 Scratch records 18-15
 - 18.6.1 CA-ADS usage 18-15
 - 18.6.2 CA-ADS/Batch considerations 18-16
- 18.7 DELETE SCRATCH 18-17
- 18.8 GET SCRATCH 18-19
- 18.9 PUT SCRATCH 18-22

18.1 Overview

CA-ADS queue and scratch management commands are used to control the allocation and access of queue and scratch records. Queue and scratch records are work records stored in the data dictionary that allow data to be passed from one CA-IDMS/DC or CA-IDMS/UCF (DC/UCF) task to another.

Note: During the execution of a CA-ADS application, each pseudo-converse is a new task.

For more information on DC/UCF tasks in the CA-ADS environment, see Chapter 4, “CA-ADS Runtime System.”

Queue records: Queue records are stored in the data dictionary queue area (DDLDCRUN). Use of queue records allows data to be passed from one DC/UCF task or batch application to another.

Scratch records: Scratch records are temporarily maintained in the data dictionary scratch area (DDLDCSCR). Under CA-ADS/Batch, scratch records can be stored in and retrieved from a scratch file allocated by the site. Use of scratch records allows data to be passed between tasks or dialogs.

Queue and scratch management commands: Queue and scratch management commands are summarized in the following table. Each command is discussed later in this section.

Type	Command	Description
Queue management	DELETE QUEUE	Deletes one or all queue records in a specified queue.
	GET QUEUE	Transfers the contents of a queue record to a specified location in a dialog's record buffers and, optionally, deletes the record from the queue.
	PUT QUEUE	Stores a queue record in the data dictionary and assigns a queue id.
Scratch management	DELETE SCRATCH	Deletes one or all scratch records associated with a specified scratch area. In CA-ADS/Batch, one or all scratch records associated with a specified scratch file are deleted.
	GET SCRATCH	Transfers the contents of a scratch record to a specified location in a dialog's record buffers and, optionally, deletes the record. In CA-ADS/Batch, the contents of a scratch record are transferred to a specified location and a scratch file is assigned to the record.
	PUT SCRATCH	Stores or replaces a scratch record in the data dictionary and assigns a scratch area id. In CA-ADS/Batch, a scratch record is stored or replaced in the scratch file and assigned a scratch area id.

18.2 Queue records

Overview: Queue records are available to all tasks running under DC/UCF, as well as to batch programs. Records in a queue established by one task are available to subsequent tasks running on the same logical terminal, or to concurrent or subsequent tasks running on any other terminal. Queue records are saved across system shutdowns and are recovered across a system crash.

Because queue records are available to concurrent tasks running on other terminals, the records can be used to pass data from one application to another. Additionally, queue records provide a convenient means of storing data for subsequent processing.

Storing a queue record: A queue record is stored in the data dictionary as a member occurrence in a set owned by a queue header record. All records associated with a particular queue header are referred to collectively as a queue. The queue is identified by a queue id. Requests to access a queue record can use the queue id to specify the queue in which the object record participates. If a request to store a queue record specifies an unknown queue id, a queue is created with the specified id.

When a queue record is stored, DC/UCF can return a queue record identifier to a specified location in a dialog's record buffers. The queue record identifier can then be used to access the queue record.

Currencies: The CA-ADS runtime system maintains currencies for each queue accessed by a task. If concurrently executing tasks access the same queue, each task has its own queue currency. A request for a particular queue record can identify the record by the queue id, by the queue record id, by the position of the record within the queue, or by the relationship of the object record to the record that is current of queue for the requesting task.

Queue records remain in the data dictionary until explicitly deleted or until the retention period specified for the queue has expired. When all records associated with a given queue header have been deleted, the header record is also deleted and the queue no longer exists.

Considerations

- An exclusive lock is placed on a queue record when the record is retrieved or stored, thereby preventing concurrently executing tasks from accessing the same record. Queue record locks are released when the task terminates or when a COMMIT command with the TASK keyword is executed.
- For information about the COMMIT command, see Chapter 16, "Database Access Commands."

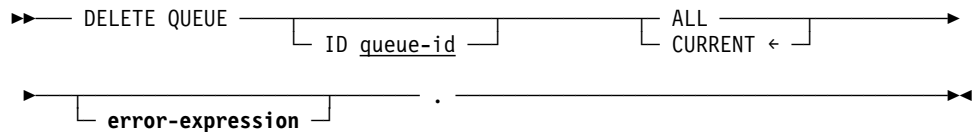
Because no other task can access a locked queue record, a concurrently executing task that attempts to access the record must wait until the lock is released. To minimize such waits, queue access should be as brief as possible.

- Queue currencies are not saved when a task terminates. Each task must establish its own currencies. The following considerations apply:
 - Queue currencies are lost each time a `DISPLAY` command is executed.
 - Queue currencies are lost across a system shutdown or a system crash.
- All queue management command clauses must be coded in the same order in which they appear in the syntax.
- Queue management commands are allowed in CA-ADS/Batch only if the application is running under the central version.

18.3 DELETE QUEUE

Purpose: Deletes a queue or queue record.

Syntax:



Parameters

ID queue-id

Specifies the queue or queue record associated with *queue-id* to be deleted.

Queue-id is the name of a variable data field that contains a queue id or the 1- to 16-character id itself, enclosed in single quotation marks.

If *queue-id* is not specified, a null queue id (that is, 16 blanks) is assumed.

ALL

Deletes all records, including the queue header record, in the queue specified by *queue-id*.

CURRENT

Deletes the record that is current of queue for the requesting task.

CURRENT is the default when you specify neither CURRENT or ALL.

error-expression

Specifies the status codes that are returned to the dialog.

►► For information about error expressions, see Chapter 10, “Error Handling.”

Usage

Considerations: If autostatus is not in use, a dialog's error-status field indicates the outcome of a DELETE QUEUE command:

Status Code	Meaning
0000	The request was executed successfully
4404	The requested header record cannot be found
4405	The requested queue record cannot be found
4406	Currency was not established for the object queue record
4407	An I/O error occurred during processing
4431	The CA-ADS internal parameter list was invalid

►► The autostatus facility is described in Chapter 10, “Error Handling.”

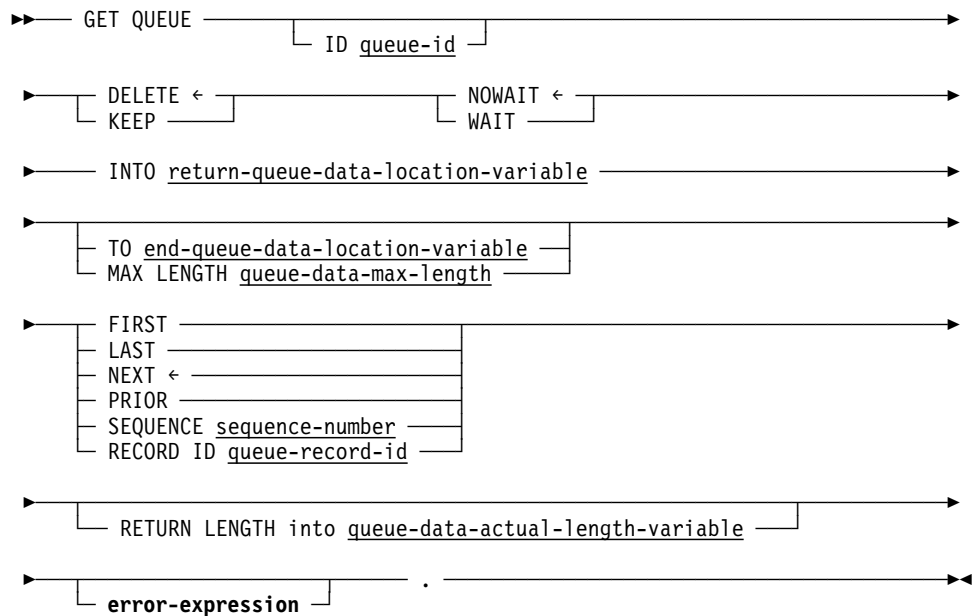
Example: The following example illustrates the use of the DELETE QUEUE command to delete the current record from queue CUSTQ:

```
DELETE QUEUE ID 'CUSTQ'.
```

18.4 GET QUEUE

Purpose: Transfers the contents of a queue record to a specified location in a dialog's record buffers.

Syntax:



Parameters

ID queue-id

Specifies the *queue-id* to be retrieved.

Queue-id is either the name of a variable data field that contains a queue id or the 1- to 16-character queue id itself, enclosed in single quotation marks.

If *queue-id* is not specified, a null queue id (that is, 16 blanks) is assumed.

DELETE

Deletes the record from the queue after it is passed to the requesting task. If the record is truncated, the truncated data may be lost permanently.

DELETE is the default when you specify neither DELETE or KEEP.

KEEP

Retains the record in the queue after it is passed to the requesting task.

NOWAIT

Continues task execution in the event of a nonexistent queue. NOWAIT is the default when you specify neither NOWAIT or WAIT.

WAIT

Suspends task execution until the requested queue exists.

INTO return-queue-data-location-variable

Specifies the location to which the requested queue record is transferred.

Return-queue-data-location-variable is the name of a variable data field in the dialog's record buffers.

TO end-queue-data-location-variable

Specifies the end of the buffer area allocated for the requested queue record.

End-queue-location-variable is either the name of a dummy byte field or the name of a variable data field that contains a data item not associated with the requested queue record. The field specified by *end-queue-data-location* must immediately follow the last byte of the buffer area allocated for the requested queue record.

MAX LENGTH queue-data-max-length

Specifies the length of the buffer area allocated for the requested queue record.

Queue-data-max-length is either the name of a variable data field that contains the length of the buffer area allocated for the requested queue record or the length itself, expressed as a numeric constant.

If neither TO *end-queue-data-location-variable* nor MAX LENGTH *queue-data-max-length* is specified, the length of the location is the length of *return-queue-data-location-variable*.

FIRST

Obtains the first record in the queue that is specified by *queue-id*.

LAST

Obtains the last record in the queue that is specified by *queue-id*.

NEXT

Obtains the record that follows the current record of the queue specified by *queue-id*.

NEXT is the default when you specify no other queue record to be obtained.

If currency is not established, NEXT is equivalent to FIRST.

PRIOR

Obtains the record that precedes the current record in the queue specified by *queue-id*.

If currency is not established, PRIOR is equivalent to LAST.

SEQUENCE sequence-number

Obtains the *n*th record in the queue specified by *queue-id*.

Sequence-number is either the name of a variable data field that contains the sequence number or the sequence number itself, expressed as a numeric constant.

RECORD ID queue-record-id

Obtains the record identified by *queue-record-id*.

Queue-record-id is either the name of a numeric variable data field that contains the system-assigned queue record id or the queue record id itself, expressed as a numeric constant.

Queue-record-id cannot be a doubleword binary field. The runtime system converts the queue record id to a binary fullword for internal storage.

RETURN LENGTH into *queue-data-actual-length-variable*

Returns the untruncated length of the obtained queue record to the location specified by *queue-data-actual-length-variable*.

Queue-data-actual-length-variable is the name of a numeric field in the dialog's record buffers.

error-expression

Specifies the status codes that are returned to the dialog.

►► For more information about error expressions, see Chapter 10, “Error Handling.”

Usage:

Considerations

- If the queue record is larger than the allocated buffer area, the record is truncated as necessary. Deletion of the record from the queue after the transfer is complete can be specified.
- If autostatus is not in use, a dialog's error-status field indicates the outcome of a GET QUEUE command:

Status Code	Meaning
0000	The request was executed successfully
4404	The requested header record cannot be found
4405	The requested queue record cannot be found
4407	An I/O error occurred during processing
4419	The dialog's storage location is too small for the requested queue record. The record was truncated accordingly
4431	The CA-ADS internal parameter list was invalid
4432	The derived length of the queue record data area is negative

►► The autostatus facility is described in Chapter 10, “Error Handling.”

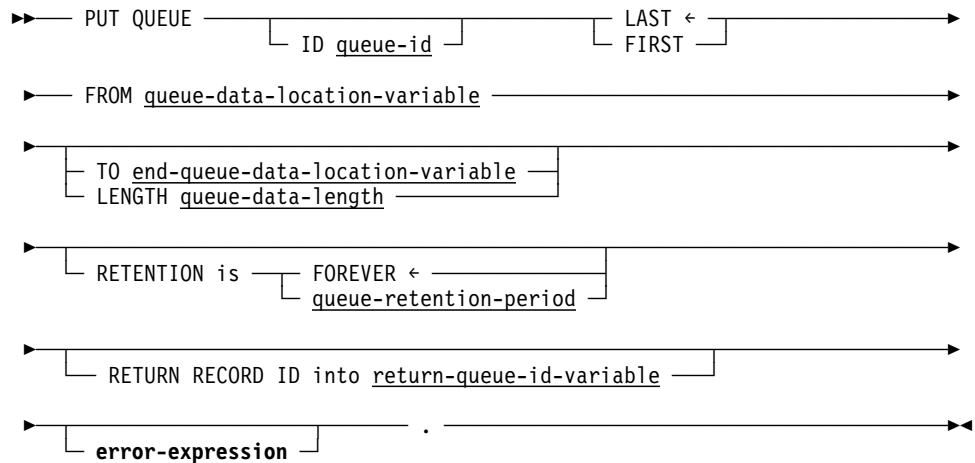
Example: The following example illustrates the use of the GET QUEUE command. The data in the last record in queue CUSTQ is transferred to the location in the dialog's record buffers identified by CUSTWORK. The record is deleted from the queue:

```
GET QUEUE ID 'CUSTQ' INTO CUSTWORK MAX LENGTH REC-LENGTH LAST.
```

18.5 PUT QUEUE

Purpose: Stores a queue record in the data dictionary.

Syntax:



Parameters

ID queue-id

Stores a record in the queue identified by *queue-id*.

Queue-id is either the name of a variable data field that contains a queue id or the 1- to 16-character queue id itself, enclosed in single quotation marks.

If *queue-id* is not specified, a null queue id (that is, 16 blanks) is assumed.

LAST

Stores a record at the end of the queue.

LAST is the default when you specify neither LAST or FIRST.

FIRST

Stores a record at the beginning of the queue.

FROM queue-data-location-variable

Specifies the location of the data to be stored in the queue record.

Queue-data-location-variable is the name of a variable data field in the dialog's record buffers.

TO end-queue-data-location-variable

Specifies the end of the buffer area that contains the queue record data.

End-queue-data-location-variable is the name of a variable data field that contains a data item not associated with the queue record data.

The field specified by *end-queue-data-location-variable* must immediately follow the last byte of the buffer area that contains the queue record data.

LENGTH queue-data-length

Specifies the length, to be specified in bytes, of the buffer area that contains the data to be stored in the queue record.

Queue-data-length is either the name of a variable data field that contains the length or the length itself, expressed as a numeric constant.

If neither *TO end-queue-data-location-variable* nor **LENGTH *queue-data-length*** is specified, the length of the location is the length of *queue-data-location-variable*.

RETENTION

Introduces the number of days, in the range 0 through 255, that the queue is to be retained.

A retention period of 255 is equivalent to FOREVER.

FOREVER

Retains the queue until all queue records associated with the queue are explicitly deleted.

FOREVER is the default when the queue's retention period is not otherwise specified.

queue-retention-period

The name of a variable data field that contains the retention period or the retention period itself, expressed as a numeric constant.

RETURN RECORD ID into return-queue-id-variable

Returns a system-assigned queue record id to the location specified by *return-queue-id-variable*.

The queue record id is returned as a binary fullword and is converted, as appropriate, when it is moved to *return-queue-id-variable*

Return-queue-id-variable is the name of a numeric variable data field in the dialog's record buffers.

Return-queue-id-variable cannot be a doubleword binary field. The system-assigned queue record id can subsequently be used to retrieve or delete the associated queue record.

error-expression

Specifies the status codes that are returned to the dialog.

►► For more information about error expressions, see Chapter 10, "Error Handling."

Usage

Considerations: If autostatus is not in use, a dialog's error-status field indicates the outcome of a PUT QUEUE command:

Status Code	Meaning
0000	The request was executed successfully
4407	The queue upper limit has been reached or an I/O error occurred during processing
4431	The CA-ADS internal parameter list was invalid

►► The autostatus facility is described in Chapter 10, “Error Handling.”

Example: The following example illustrates the use of the PUT QUEUE command to store the data in CUSTWORK in a queue record associated with queue CUSTQ:

```
PUT QUEUE ID 'CUSTQ' FROM CUSTWORK LENGTH REC-LENGTH  
RETURN RECORD ID INTO REC-ID.
```

18.6 Scratch records

Scratch records allow a task to pass information to subsequent tasks, thereby providing data continuity among tasks. The scratch records are used only for temporary storage of data and are not saved across a system shutdown or a system crash.

Scratch area id: A scratch area is identified by an eight-character name. Requests to access a scratch record can use the scratch area id to specify the area with which the object record is associated. If a request to store a scratch record specifies an unknown scratch area id, a scratch area is created with the specified id.

Scratch records are also assigned numeric identifiers either by the application developer or by the system. Records in a scratch area are arranged sequentially in ascending order, according to the value of the scratch record identification. System-assigned identifiers are sequenced last in a scratch area.

All scratch management command clauses must be coded in the same order in which they appear in the syntax.

18.6.1 CA-ADS usage

Scratch records are common to all tasks running on the same logical terminal. The records stored by one task are available to subsequent tasks running on the same terminal.

A request to store a scratch record places a record of the requested length in the data dictionary. A database key pointer to the scratch record is placed in a scratch area associated with the requesting task. Scratch records remain in the data dictionary until explicitly deleted, until a signoff from DC/UCF occurs, or until the system is shut down or crashes.

Currencies are maintained for each scratch area associated with a task. Scratch area currencies are passed from one task to the next. A request for a particular scratch record can identify the record by the scratch area id, by the scratch record id, by the position of the record within the scratch area, or by the relationship of the object record to the record that is current of the scratch area.

Considerations

- Scratch records associated with one terminal are not available to tasks associated with other terminals.
- Any number of scratch records can be associated with a single scratch area, and any number of scratch areas can be associated with a task.
- When all records associated with a given scratch area have been deleted, the scratch area is also deleted.
- During the execution of a CA-ADS application, each pseudo-converse is a new task.

►► For more information on DC/UCF tasks in the CA-ADS environment, see Chapter 4, “CA-ADS Runtime System.”

18.6.2 CA-ADS/Batch considerations

Information can be written to temporary scratch files at dialog execution time and passed between dialogs within the same job step in a given CA-ADS/Batch application. A request to store a scratch record places a record of the requested length in a temporary work file. Records can be accessed in any order from this file. The scratch file need not be defined to the data dictionary.

Scratch records remain in the temporary file until they are explicitly deleted, until the job step is completed, until a signoff occurs, or if the system is shut down or crashes.

Using scratch files: To use scratch files:

1. **Include process-language SCRATCH statements in dialog process modules.** At dialog execution time, these statements store, retrieve, and delete scratch records.

Syntax for SCRATCH statements in CA-ADS/Batch dialogs is the same as for CA-ADS dialogs. SCRATCH statement syntax is presented later in this section.

2. **Define the external name for a scratch file** in the DMCL module for scratch (SCRDMCL).
3. **Initialize a data set for the scratch file the first time the file is used** by using the FORMAT utility.

►► For information about FORMAT job control language statements for initializing a scratch file, see Appendix D, “Application and Dialog Utilities.”

4. **Include FORMAT job control language statements** immediately before control statements for the CA-ADS/Batch application.
5. **Specify the ddname/filename for the scratch file** in the CA-ADS/Batch job to make the initialized scratch file available to the application.

►► For more information on the FORMAT utility and its input parameters, refer to *CA-IDMS Utilities*.

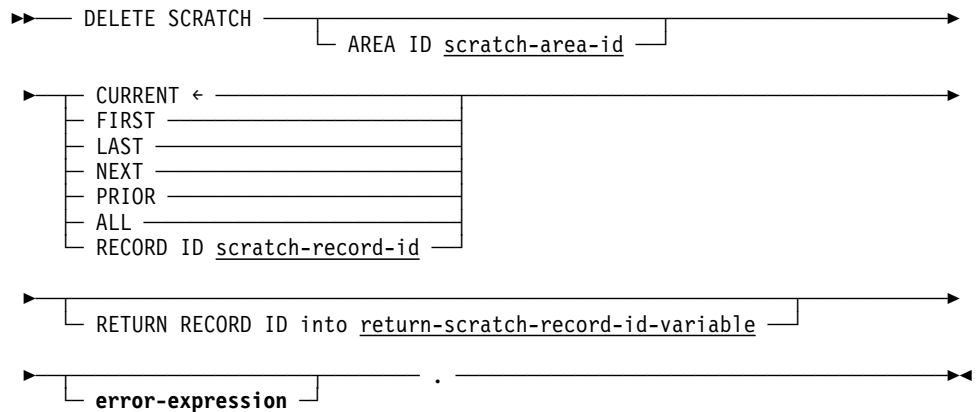
Considerations

- The scratch file cannot be used to communicate between CA-ADS/Batch job steps.
- The same scratch file can be used in several CA-ADS/Batch job steps without reinitializing the file. A PUT SCRATCH command must be used before any GET SCRATCH commands.
- Central version must be used to run CA-ADS/Batch.

18.7 DELETE SCRATCH

Purpose: Deletes one or all records associated with a particular scratch area id.

Syntax:



Parameters

AREA ID scratch-area-id

Specifies the area in the data dictionary scratch area to be deleted.

Scratch-area-id is either the name of a variable data field that contains a scratch area id or the 1- to 8-character scratch area id itself, enclosed in single quotation marks.

If *scratch-area-id* is not specified, a null scratch area id (that is, eight blanks) is assumed.

CURRENT

Deletes the record that is current of the scratch area specified by *scratch-area-id*.

CURRENT is the default when you specify no other scratch record to be deleted.

FIRST

Deletes the first record in the scratch area specified by *scratch-area-id*.

LAST

Deletes the last record in the scratch area specified by *scratch-area-id*.

NEXT

Deletes the record that follows the current record of the scratch area specified by *scratch-area-id*.

If currency is not established, NEXT is equivalent to FIRST.

PRIOR

Deletes the record that precedes the current record of the scratch area specified by *scratch-area-id*.

If currency is not established, PRIOR is equivalent to LAST.

ALL

Deletes all records in the scratch area specified by *scratch-area-id*.

RECORD ID scratch-record-id

Deletes the record identified by *scratch-record-id*.

Scratch-record-id is either the name of a variable data field that contains the scratch record id or the scratch record id itself, expressed as a numeric constant.

RETURN RECORD ID into return-scratch-record-id-variable

Returns the id of the last scratch record deleted to the location specified by *return-scratch-record-id-variable*.

Return-scratch-record-id-variable is the name of a numeric variable data field in the dialog's record buffers.

Return-scratch-record-id-variable cannot be a doubleword binary field.

error-expression

Specifies the status codes that are returned to the dialog.

►► Error expressions are described in Chapter 10, "Error Handling."

Usage:

Considerations If autostatus is not in use, a dialog's error-status field indicates the outcome of a DELETE SCRATCH command:

Status Code	Meaning
0000	The request was executed successfully.
4303	The requested scratch area cannot be found.
4305	The requested scratch record cannot be found.
4307	An I/O error occurred during processing.
4331	The CA-ADS internal parameter list is invalid.

►► The autostatus facility is described in Chapter 10, "Error Handling."

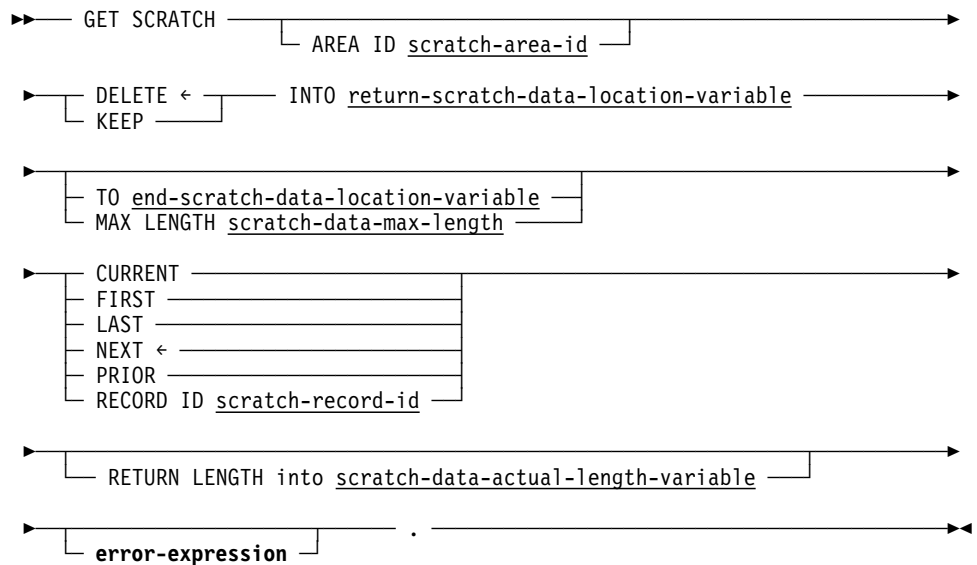
Example: The following example illustrates the use of the DELETE SCRATCH command to delete all of the records in scratch area CUSTAREA:

```
DELETE SCRATCH AREA ID 'CUSTAREA' ALL.
```


18.8 GET SCRATCH

Purpose: Transfers the contents of a scratch record to a specified location in a dialog's record buffers.

Syntax:



Parameters

AREA ID scratch-area-id

Specifies an area in the scratch area to be retrieved.

Scratch-area-id is either the name of a variable data field that contains a scratch area id or the 1- to 8-character scratch area id itself, enclosed in single quotation marks.

If *scratch-area-id* is not specified, a null scratch area id (that is, eight blanks) is assumed.

DELETE

Deletes the record from the scratch area after it is passed to the requesting task.

If the record is truncated, the truncated data may be lost permanently.

DELETE is the default when you specify neither DELETE or KEEP.

KEEP

Retains the record in the scratch area after it is passed to the requesting task.

INTO return-scratch-data-location-variable

Specifies the location to which the requested scratch record is transferred.

Return-scratch-data-location-variable is the name of a variable data field in the dialog's record buffers.

TO end-scratch-data-location-variable

Specifies the end of the buffer area allocated for the requested scratch record.

End-scratch-data-location-variable is the name of a dummy byte field or the name of a variable data field that contains a data item not associated with the requested scratch record.

The field specified by *end-scratch-data-location-variable* must immediately follow the last byte of the buffer area allocated for the requested scratch record.

MAX LENGTH scratch-data-max-length

Specifies the length of the buffer area allocated for the requested scratch record.

Scratch-data-max-length is the name of a variable data field that contains the length or the length itself, expressed as a numeric constant.

If neither TO *end-scratch-data-location-variable* nor MAX LENGTH *scratch-data-max-length* is specified, the length of the location is the length of *return-scratch-data-location-variable*.

CURRENT

Obtains the record that is current of the scratch area specified by *scratch-area-id*.

FIRST

Obtains the first record in the scratch area specified by *scratch-area-id*.

LAST

Obtains the last record in the scratch area specified by *scratch-area-id*.

NEXT

Obtains the record that follows the current record of the scratch area specified by *scratch-area-id*.

NEXT is the default when you specify no other scratch record to be obtained.

If currency is not established, NEXT is equivalent to FIRST.

PRIOR

Obtains the record that precedes the current record of the scratch area specified by *scratch-area-id*.

If currency is not established, PRIOR is equivalent to LAST.

RECORD ID scratch-record-id

Obtains the record identified by *scratch-record-id*.

Scratch-record-id is either the name of a variable data field that contains the scratch record id or the scratch record id itself, expressed as a numeric constant.

RETURN LENGTH into scratch-data-actual-length-variable

Returns the untruncated length of the obtained scratch record to the location specified by *scratch-data-actual-length-variable*.

Scratch-data-actual-length-variable is the name of a numeric field in the dialog's record buffers.

error-expression

Specifies the status codes that are returned to the dialog.

►► Error expressions are described in Chapter 10, “Error Handling.”

Usage:

Considerations: If the scratch record is larger than the allocated buffer area, the record is truncated as necessary.

If autostatus is not in use, a dialog's error-status field indicates the outcome of a GET SCRATCH command:

Status Code	Meaning
0000	The request was executed successfully
4303	The requested scratch area cannot be found
4305	The requested scratch record cannot be found
4307	An I/O error occurred during processing
4319	The dialog's storage location is too small for the requested scratch record. The record was truncated accordingly
4331	The CA-ADS internal parameter list was invalid
4332	The derived length of the scratch record data area is negative.

►► The autostatus facility is described in Chapter 10, “Error Handling.”

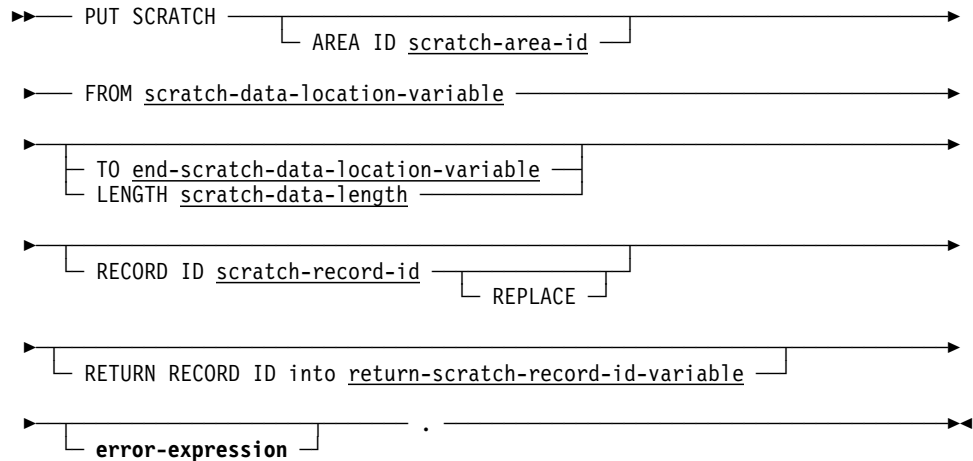
Example: The following example illustrates the use of the GET SCRATCH command to copy the last record in scratch area CUSTAREA to a location in the dialog's record buffers identified by CUSTWORK. The record is retained in the scratch area for later access:

```
GET SCRATCH AREA ID 'CUSTAREA' KEEP LAST  
INTO CUSTWORK MAX LENGTH REC-LENGTH.
```

18.9 PUT SCRATCH

Purpose: Stores or replaces a scratch record in the scratch area.

Syntax:



Parameters

AREA ID scratch-area-id

Specifies the area in the scratch area where the record will be stored.

Scratch-area-id is either the name of a variable data field that contains a scratch area id or the 1- to 8-character scratch area id itself, enclosed in single quotation marks.

If *scratch-area-id* is not specified, a null scratch area id (that is, eight blanks) is assumed.

FROM scratch-data-location-variable

Specifies the location of the data to be stored in the queue record.

Scratch-data-location-variable is the name of a variable data field in the dialog's record buffers.

TO end-scratch-data-location-variable

Specifies the end of the buffer area that contains the data to be stored in the scratch record.

End-scratch-data-location-variable is either the name of a dummy byte field or the name of a variable data field that contains a data item not associated with the scratch record data.

The field specified by *end-scratch-data-location-variable* must immediately follow the last byte of the buffer area that contains the scratch record data.

LENGTH scratch-data-length

Specifies the length, to be specified in bytes, of the buffer area that contains the data to be stored in the scratch record.

Scratch-data-length is either the name of a variable data field that contains the length or the length itself, expressed as a numeric constant.

If neither *TO end-scratch-data-location-variable* nor *LENGTH scratch-data-length* is specified, the length of the location is the length of *scratch-data-location-variable*.

RECORD ID scratch-record-id

Assigns an id to the scratch record being stored.

Scratch-record-id is either the name of a variable data field that contains the scratch record id or the scratch record id itself, expressed as a numeric constant.

The scratch record id can subsequently be used to retrieve or delete the associated scratch record.

The scratch record id is stored as a binary fullword.

REPLACE

Replaces the scratch record identified by *scratch-record-id* with the scratch record being stored.

RETURN RECORD ID into return-scratch-record-id-variable

Returns a system-assigned scratch record id to the location specified by *return-scratch-record-id-variable*.

Return-scratch-record-id-variable is the name of a variable data field in the dialog's record buffers.

Return-scratch-record-id-variable cannot be defined as a doubleword binary field.

The scratch record id can subsequently be used to retrieve or delete the associated scratch record.

The system assigns a scratch record id if one is not specified in the **RECORD ID** parameter.

error-expression

Specifies the status codes that are returned to the dialog.

►► Error expressions are described in Chapter 10, "Error Handling."

Usage:

Considerations: If autostatus is not in use, a dialog's error-status field indicates the outcome of a PUT SCRATCH command:

Status Code	Meaning
0000	The request to add a scratch record was executed successfully
4307	An I/O error occurred during processing
4317	The request to replace a scratch record was executed successfully
4322	The request to store a scratch record cannot be executed because the scratch record id already exists within the scratch area and the REPLACE option was not specified
4331	The CA-ADS internal parameter list was invalid
4332	The derived length of the scratch record data location is negative

►► The autostatus facility is described in Chapter 10, “Error Handling.”

Example: The following example illustrates the use of the PUT SCRATCH command:

```
PUT SCRATCH AREA ID 'CUSTAREA' FROM CUSTWORK LENGTH REC-LENGTH  
RETURN RECORD ID INTO REC-ID.
```

Chapter 19. Subroutine Control Commands

19.1 Overview	19-3
19.2 CALL	19-4
19.3 DEFINE	19-5
19.4 GOBACK	19-6

19.1 Overview

CA-ADS subroutine control commands are used to define and call subroutines within a process.

The subroutine control commands are listed in the following table. Each command is discussed later in this section.

Subroutine control commands

Command	Description
CALL	Passes control to a predefined subroutine
DEFINE	Establishes an entry point for a subroutine and defines subroutine processing
GOBACK	Terminates subroutine processing and returns control to the command following the associated CALL command

19.2 CALL

Purpose: Passes control to a predetermined subroutine.

Syntax:

► CALL subroutine-name . ◄

Parameter

subroutine-name

Specifies the 1- to 8-character name of the subroutine to which control is passed. The subroutine name is defined by the DEFINE command, described below.

Usage: When the CA-ADS runtime system encounters a CALL command, processing control passes to the beginning of the named subroutine. Processing continues through the subroutine until CA-ADS encounters a GOBACK command, or a control command

► For more information about GOBACK, see 19.4, “GOBACK” later in this chapter.

For more information about control commands, in Chapter 15, “Control Commands.”

If no GOBACK or control command occurs before the end of the subroutine, the runtime system automatically returns control to the command that immediately follows the associated CALL command.

Considerations

- A CALL statement can occur in the body of a process or within a subroutine definition.
- Subroutine calls can be nested up to ten levels.
- The called subroutine must be defined by using the DEFINE command, described later in this section, and must be coded later in the process than the CALL command.

19.3 DEFINE

Purpose: Establishes an entry point for a subroutine and to define the subroutine processing. At runtime, a subroutine is executed when it is named in a CALL statement.

Syntax:

► — DEFINE subroutine subroutine-name — . — command statement. — ◄

Parameters

subroutine-name

Specifies the 1- to 8-character name of the subroutine being defined.

Subroutine-name must be unique within the process.

command-statement

Specifies the process commands that define the subroutine.

Command-statement can be any process command statement except DEFINE SUBROUTINE.

Usage

Considerations

- Each command statement must be terminated with a period (.).
- Any number of subroutine definitions can be coded at the end of a process.
- A subroutine definition is terminated by the occurrence of another subroutine definition or by the end of the process code.
- DEFINE is the only process command that can follow a subroutine definition.
- Subroutine order and physical placement within the process code are important. The physical placement of multiple subroutines depends on the code within each subroutine. All called subroutines must be physically coded lower than the calling subroutine:

```
Example:
CALL SUBROUTINE-A.
....
CALL SUBROUTINE-B.
....
DEFINE SUBROUTINE-A.
  (within this code which is subroutine-d)
DEFINE SUBROUTINE-D.
  (within this code which is subroutine-b)
DEFINE SUBROUTINE-B.
```

19.4 GOBACK

Purpose: Terminates subroutine processing.

Syntax:

►► GOBACK — . —————►◄

Usage: At run time, GOBACK returns processing control to the command following the CALL that passed control to the subroutine.

Considerations

- A GOBACK command can be coded wherever logically appropriate within the body of a subroutine.
- A GOBACK command is automatically generated by CA-ADS to ensure that GOBACK is the last command in the subroutine.

Example: The following example uses the CALL, DEFINE, and GOBACK commands to illustrate the use of a subroutine within a process:

```
FIND CALC CUSTOMER.
IF DB-REC-NOT-FOUND
THEN
  DO.
    STORE CUSTOMER.
    CALL UPDMAIL.
    DISPLAY MSG TEXT 'CUSTOMER ADDED'.
  END.
ELSE
  DO.
    MODIFY CUSTOMER.
    CALL UPDMAIL.
    DISPLAY MESSAGE TEXT 'CUSTOMER CHANGED'.
  END.
DEFINE SUBROUTINE UPDMAIL.
  MOVE 1 TO SB.
  WHILE SB LE 3
  REPEAT.
    MOVE CUST-INT(SB) TO MAIL-INT.
    FIND CALC MAILIST.
    CONNECT CUSTOMER TO MAILIST.
    ADD 1 TO SB.
  END.
GOBACK.
```

Chapter 20. Utility Commands

20.1 Overview	20-3
20.2 ABORT	20-4
20.3 ACCEPT	20-8
20.4 INITIALIZE RECORDS	20-10
20.5 SNAP	20-11
20.6 TRACE	20-13
20.7 WRITE PRINTER	20-14
20.8 WRITE TO LOG/OPERATOR	20-18

20.1 Overview

CA-ADS utility commands are used to reinitialize record buffers, transmit data to be printed, and provide information about the current task.

The utility commands are summarized in the following table.

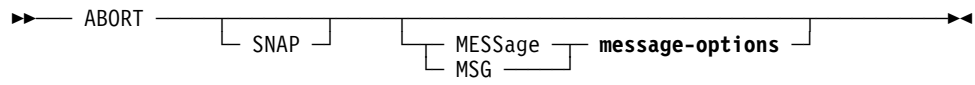
Summary of utility commands

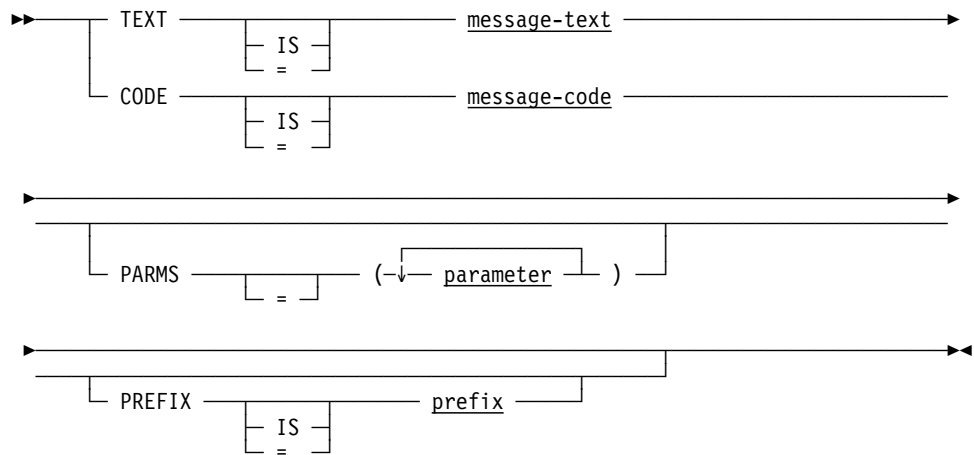
Command	Description
ABORT	Abnormally terminates an application
ACCEPT	Retrieves task-related information
INITIALIZE RECORDS	Reinitializes dialog record buffers
SNAP	Requests a snap dump of selected memory areas
WRITE PRINTER	Transmits data from a dialog to a CA-IDMS/DC or CA-IDMS/UCF print queue
WRITE TO LOG/OPERATOR	Sends a message to the log file or to the operator's console (CA-ADS/Batch only)

20.2 ABORT

Purpose: Terminates the execution of the current task.

Syntax:



Expansion of message-options**Parameters****SNAP**

Writes a formatted snap dump to the CA-IDMS/DC or CA-IDMS/UCF (DC/UCF) log. Snap dumps can be printed by means of the PRINT LOG utility.

►► For information on PRINT LOG, refer to *CA-IDMS Utilities*.

MESSAge

Specifies a message to be displayed on the Dialog Abort Information screen and written to the system log. If a MESSAGE clause is not specified, system message DC174020 is used (text of this message can be changed by using IDD):

ADS/ON-LINE ABORT. USER SPECIFIED ABORT WITH NO MESSAGE CODE/TEXT

MSG can be used in place of MESSAGE.

►► For more information about altering message text using IDD, refer to the *IDD DDDL Reference*, under the MESSAGE command.

TEXT IS message-text

Specifies the text of a message to be displayed in an online map's message field or sent to a batch application and a system log file.

This can be either the name of a variable data field containing the message text or the text string itself, enclosed in single quotation marks.

The text string can contain up to 240 displayable characters.

CODE IS message-code

Specifies the message dictionary code of a message to be displayed in an online map's message field or sent to the log file in a batch application.

This can be either the name of a variable data field that contains the message code or the 6-digit code itself, expressed as a numeric literal.

In a batch application, the message is also sent to the operator, if directed by the destination specified in the dictionary.

PARMS = parameter

Introduces a replacement parameter for each variable field in the stored message identified by *message-code*. The parameter can be either the name of an EBCDIC or unsigned zoned decimal variable data field that contains the parameter value or the actual parameter value, enclosed in single quotation marks.

Up to nine replacement parameters can be specified for a message. Multiple parameters must be specified in the order in which they are numbered and separated by blanks or commas.

The parameter value must contain displayable characters. At run time, each variable data field in a stored message expands or contracts to accommodate the size of its replacement parameter. A replacement parameter can be a maximum of 240 bytes.

PREFIX IS prefix

Overrides the default prefix of a dialog and a map. *Prefix* specifies an EBCDIC or unsigned zoned decimal variable data field that contains a 2-character prefix or the 2-character prefix itself, enclosed in single quotation marks

Usage*Considerations*

- When a dialog issues an ABORT command, CA-ADS abnormally terminates the current task and returns control to DC/UCF. A snap dump of all memory areas maintained for the current CA-ADS runtime session at the time of the abort can be requested.
- The CA-ADS runtime system provides a diagnostic screen that displays information about an abnormally terminated dialog. The diagnostic screen is enabled for an installation by means of the DIAGNOSTIC SCREEN clause of the system generation ADSO statement.
 - ▶▶ For more information on the ADSO statement, refer to *CA-IDMS System Generation*.

If the diagnostic screen is not enabled when an ABORT command is issued, a system error message (DC466019) is displayed. If a message code is specified in the ABORT command and the dictionary message specifies a destination of log, the message is also sent to the system log.

▶▶ For more information on the Dialog Abort Information screen, see Chapter 4, “CA-ADS Runtime System.”

- Up to nine replacement parameters can be specified for a message.
- Multiple message parameters must be separated by blanks or commas.

- Message parameters must be specified in the order in which they occur in the stored message.
- Within the message definition in the dictionary, symbolic parameters are identified by an ampersand (&) followed by a two-digit numeric identifier. These identifiers can appear in any order. The position of the replacement values in the PARMs parameter must correspond directly to the two-digit numeric identifiers in the message; the first value corresponds to &01, the second to &02, and so forth. For example, assume that the stored message text is as follows:

THIS IS TEXT &01 AND &03 OR &02

The PARMs parameter reads PARMs=('A','B','C'). The resulting text would read as follows:

THIS IS TEXT A AND C OR B

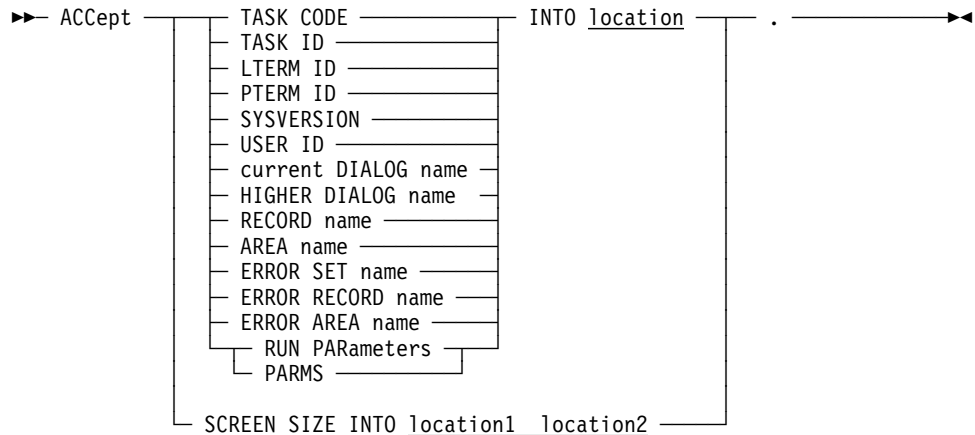
Example: The following example illustrates the use of the ABORT command:

```
ADD ACC-BAL TO TOT-BAL.
ADD 1 TO CONTROL-CTR.
IF CONTROL-CTR < 100
THEN
    INVOKE 'CEXDR008'.
ELSE
    ABORT SNAP MSG TEXT CUST-NUM.
```

20.3 ACCEPT

Purpose: Retrieves information about the current task and dialog. (In ADS/Batch, ACCEPT is used to accept runtime parameters into a storage location.)

Syntax:



Parameters

TASK CODE

Retrieves the eight-character code used to invoke the current task.

TASK ID

Retrieves the DC/UCF-assigned task identification number. The task id is a unique sequence number stored in a binary fullword field. The task id is zero when DC/UCF is started and is incremented by one for each new task added to the system.

LTERM ID

Retrieves the eight-character identification of the logical terminal associated with the current task.

PTERM ID

Retrieves the eight-character identification of the physical terminal associated with the current task.

SYSVERSION

Retrieves the version number, in the range 0 to 32767, of the DC/UCF system currently in use. The version number is stored in a binary halfword field.

USER ID

Retrieves the user identification.

In CA-ADS, the 32-character identification of the user signed on to the logical terminal associated with the current task is retrieved. If no user is signed on, a null user id (that is, 32 blanks) is returned.

In CA-ADS/Batch, the user identification specified in the USER (REQUESTOR) input parameter is retrieved.

current DIALOG name

Retrieves the name of the current dialog.

HIGHER DIALOG name

Retrieves the name of the dialog that is operative at the next higher level in the current application thread.

RECORD name

Retrieves the name of the record that is current of run unit for the issuing dialog.

AREA name

Retrieves the name of the area that is current of area for the issuing dialog.

ERROR SET name

Retrieves the name of the last set involved in an operation that resulted in an error condition.

ERROR RECORD name

Retrieves the name of the last record involved in an operation that resulted in an error condition.

ERROR AREA name

Retrieves the name of the last area involved in an operation that resulted in an error condition.

RUN PARAmeters

(CA-ADS/Batch only) Retrieves runtime parameters, which are specified in the JCL PARM parameter (OS/390, VSE/ESA Release 2.1, and VM/ESA) or in a JOB VARIABLE statement (BS2000/OSD). If no runtime parameters are specified in the JCL, the storage location is blank filled.

PARMS can be used in place of RUN PAR.

INTO location

Specifies the location to which the information is moved.

Location is the name of a variable data field in the dialog's record buffers. The specified field must have an appropriate picture and usage for the value being retrieved.

SCREEN SIZE INTO location1 location2

Retrieves the dimensions (that is, the number of rows and columns) of the physical terminal screen associated with the current task.

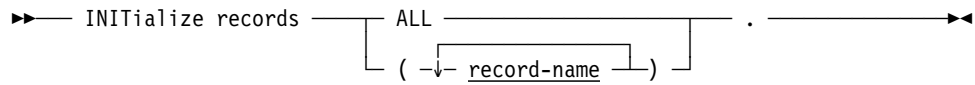
Location1 is the name of a numeric variable data field in the dialog's record buffers to which the number of rows moved.

Location2 is the name of a numeric variable data field in the dialog's record buffers to which the number of columns is moved.

20.4 INITIALIZE RECORDS

Purpose: Reinitializes one or more of a dialog's record buffers.

Syntax:



Parameters

ALL

Reinitializes the buffers for all subschema, map, and work records referenced by the issuing dialog, regardless of which dialog originally allocated the buffers.

record-name

Reinitializes the buffer for each record specified by *record-name*. The named records must be associated with the issuing dialog.

Usage:

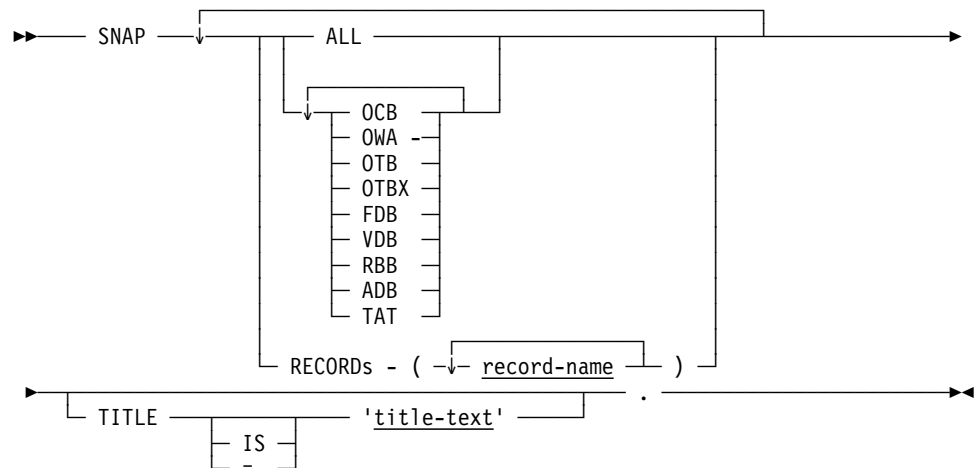
Considerations

- After execution of an INITIALIZE RECORDS command, the record elements in the specified record buffers contain their original values, as follows:
 - If the record element is defined with a VALUE IS clause, the buffer is reinitialized with the defined value.
 - If the record element definition has no VALUE IS clause, the buffer is reinitialized either with spaces (EBCDIC and DBCS fields) or with zeros of the appropriate data type (numeric fields).

20.5 SNAP

Purpose: Request a snap dump of the contents of one or more areas maintained in memory for CA-ADS. The dump produced by a SNAP command can be used to assess the use of system resources by an executing dialog.

Syntax:



Parameters

ALL

Writes all areas of memory maintained for the issuing dialog to the SNAP dump.

OCB

Keyword which requests the CA-ADS control block. The OCB contains CA-ADS system parameters specified in the system generation ADSO statement.

OWA

Keyword which specifies the CA-ADS online work area. The OWA is maintained as a temporary storage buffer for application and dialog information used during CA-ADS runtime processing. The OWA is not maintained across tasks.

OTB

Keyword which specifies the CA-ADS terminal block. The OTB contains information about the current CA-ADS session. The OTB is maintained across tasks.

OTBX

Keyword which specifies the CA-ADS terminal block extension. The OTBX is an extension of the OTB and contains pointers to the TAT, the ADSO-APPLICATION-GLOBAL-RECORD record buffer, the RBB, and the ADB for the currently executing application. The OTBX exists only for applications defined using the application compiler (ADSA).

FDB

Keyword which specifies the fixed dialog block. The FDB is the dialog load module created by the dialog compiler (ADSC). Information in the FDB includes executable process code and parameters required to execute the dialog, and information on the maps and records associated with the dialog.

VDB

Keyword which specifies the variable dialog block. One VDB exists for each operative dialog. A VDB contains runtime variable information about a dialog, such as the status of map fields, information concerning flow of control, addresses of records used by the dialog, and the address of the executing command.

The VDB is created dynamically for the issuing dialog at runtime.

RBB

Keyword which specifies the record buffer block. The RBB contains header information and buffers for all records associated with the current application.

ADB

Keyword which specifies the application definition block. The ADB is the application load module created by the CA/ADS online application compiler. The ADB contains the application information supplied on the definition screens during an application compiler session. The ADB exists only if the application is defined using the application compiler.

TAT

Keyword which specifies the task application table. The TAT contains the names of task codes used to initiate applications and the names of the applications (ADBs) thus initiated. The TAT exists only if there are applications on the system that are defined using the application compiler.

RECORDS record-name

Includes information associated with the specified subschema, map, or work records in the SNAP dump. The information is taken from the RBB; it includes data, but no headers, from the buffers for the named records.

Record-name must be associated with the issuing dialog.

TITLE is 'title-text'

Specifies a title for the SNAP dump.

Title-text is a 1- to 90-character string enclosed in single quotation marks. The specified title is printed on the hard-copy listing of the SNAP dump.

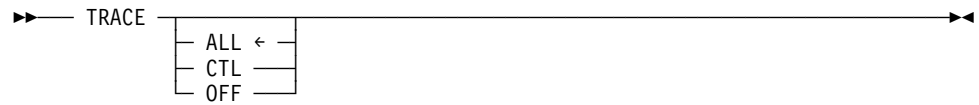
Usage: Snap dumps are written to the DC/UCF log and can be printed by using the PRINT LOG print log utility.

►► For information on PRINT LOG, refer to *CA-IDMS Utilities*.

20.6 TRACE

Purpose: Activates the CA-ADS trace facility; with the OFF parameter, deactivates the CA-ADS trace facility.

Syntax



Parameters

ALL

Writes trace records to the system log for each of the following:

- Dialog entry
- Process module entry
- Subroutine entry
- Process command execution for dialogs having symbol tables
- Database status information
- Currency save and restore operations

CTL

Writes the same trace records as ALL only for the following subset of process commands:

- Control commands
- Database commands

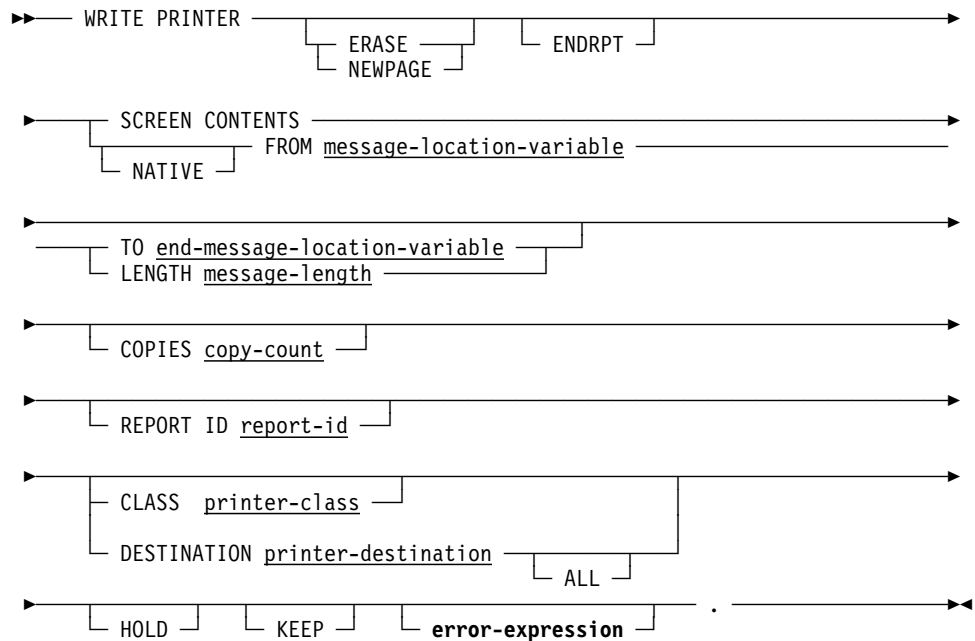
OFF

Deactivates the trace facility.

20.7 WRITE PRINTER

Purpose: Sends data from a dialog to a printer or to a file.

Syntax:



Parameters

ERASE

Specifies that the data being transmitted is to be printed on a new page.

NEWPAGE may be used in place of ERASE.

ENDRPT

Specifies that the data being transmitted is the last record of the specified report.

If ENDRPT is specified, the report is printed before the current task terminates.

SCREEN CONTENTS

Transmits the contents of the currently displayed screen to the print queue. This option is valid only for 3270-type terminals. If SCREEN CONTENTS is specified for another terminal type, an error condition results.

NATIVE

Specifies that the data stream being transmitted contains line and device control characters. If NATIVE is not specified, DC/UCF automatically inserts the necessary characters.

FROM message-location-variable

Specifies the location of the data to be transmitted to the print queue.

Message-location-variable is the name of a variable data field in the dialog's record buffers.

TO end-message-location-variable

Specifies the end of the buffer area that contains the data to be transmitted.

End-message-location-variable is the name of a dummy byte field or the name of a variable data field that contains a data item not associated with the data being transmitted.

The field specified by *end-message-location-variable* must immediately follow the last byte of the buffer area that contains the data to be transmitted.

LENGTH message-length

Specifies the length of the buffer area that contains the data to be transmitted.

Message-length is either the name of a numeric variable data field that contains the length in bytes or the length itself, in bytes, expressed as a numeric constant.

COPIES copy-count

Specifies the number of report copies to print.

Copy-count-number is either the name of a numeric variable data field that contains the copy count or the number of report copies itself, expressed as a numeric constant in the range 1 through 255.

If COPIES is not specified, the number of copies defaults to 1.

REPORT ID report-id

Specifies the report with which the transmitted data is associated. The report id must be an integer in the range 1 through 255.

Report-id is either the name of a numeric variable data field that contains the report id or the report id itself expressed as a numeric constant.

If REPORT ID is not specified, the report id defaults to 1.

CLASS printer-class

Specifies the print class, in the range 1 through 64, to which the report is assigned.

Printer-class is either The name of a numeric variable data field that contains the print class or the print class itself, expressed as a numeric constant.

If no print class is specified, the physical terminal default is used.

DESTINATION printer-destination

Specifies the printer to which the report is routed.

Printer-destination is either the name of a variable data field that contains the 1 to 8-character destination or the destination itself, enclosed in single quotation marks.

If no print destination is specified, the physical terminal default is used.

ALL

Specifies that the report is to be printed on all of the logical terminals at the specified print destination. If ALL is not specified, the report is printed on only one of the logical terminals.

HOLD

Specifies that DC/UCF is not to print the report until a system operator releases it with a DCMT VARY REPORT command.

►► For information about DCMT commands, see *CA-IDMS System Tasks and Operator Commands*.

KEEP

Specifies that each time DC/UCF finishes printing the report, the report is to be kept instead of deleted. The report can be reprinted or deleted with a DCMT VARY REPORT command.

If KEEP is not specified, the report is deleted once it is printed.

error-expression

Specifies the status codes that are returned to the dialog.

►► Error expressions are described in Chapter 10, “Error Handling.”

Usage

Definition: The WRITE PRINTER command is used to transmit data from the issuing dialog to a DC/UCF printer terminal and to initiate printing of the transmitted data. Data is passed first to a report queue maintained by DC/UCF and then to the printer.

Each line of data transmitted by a WRITE PRINTER request is considered a record. Each record is associated with a particular report in the report queue. A report consists of one or more records. The report queue can contain up to 256 active reports for any one task.

If autostatus is not in use, a dialog's error-status field indicates the outcome of a WRITE PRINTER command:

Status Code	Meaning
0000	The request was executed successfully
4807	An I/O error occurred in placing the record in the print queue
4818	The DC/UCF system has no logical terminals associated with a printer
4821	The specified printer destination is invalid
4838	The variable storage field that contains the record to be printed was not allocated
4845	The output terminal type is not correct for the WRITE PRINTER request
4846	A terminal I/O error occurred while attempting to print the contents of a screen.

- ▶▶ The autostatus facility is described under Chapter 10, “Error Handling.”

Considerations

- A report is terminated when the current run unit is terminated or when a WRITE PRINTER ENDRPT command is issued. Note that a run unit can be extended across dialogs by using the LINK command.
 - ▶▶ For more information on extending a run unit, see Chapter 4, “CA-ADS Runtime System” and Chapter 15, “Control Commands.”
- A process can contain multiple WRITE PRINTER requests, each for a different report. DC/UCF maintains the records associated with each report individually, ensuring that records associated with one report are not interspersed with records associated with other reports when the reports are printed.
- Printing is initiated either explicitly by a WRITE PRINTER request or implicitly by termination of the current task. If a task terminates abnormally, all data in the print queue is deleted, unless it was previously committed by a COMMIT TASK command.
- Each printer has one or more DC/UCF classes or destinations. The print class and destination for a report are assigned when the WRITE PRINTER command is issued for the first record in the report. The entire report is printed on the first available printer with the specified class.
- A default print class and print destination can be specified for applications defined using the application compiler. The defaults are specified on the General Options screen.
 - ▶▶ For more information, see Chapter 2, “CA-ADS Application Compiler (ADSA).”

At runtime, the defaults are stored in the AGR-PRINT-CLASS and AGR-PRINT-DESTINATION record elements of the ADSO-APPLICATION-GLOBAL-RECORD. WRITE PRINTER commands can select these defaults by specifying these record elements in the CLASS and DESTINATION parameters.

- ▶▶ For more information on the ADSO-APPLICATION-GLOBAL-RECORD, see Appendix A, “System Records.”

20.8 WRITE TO LOG/OPERATOR

Purpose: Sends messages to the log file or, in the batch environment, to the log file and the operator's console.

Syntax:

```

➤➤ WRITE — to — [ LOG — ] [ MESSAGE — ] message-options — . ➤➤
                  [ OPERator ] [ MSG ]

```

Expansion of message-options

```

➤➤ [ TEXT — ] [ IS — ] message-text — ➤➤
    [ CODE — ] [ IS — ] message-code — ➤➤
    [ PARS — ] [ = — ] ( — parameter — ) — ➤➤
    [ PREFIX — ] [ IS — ] prefix — ➤➤

```

Parameters

LOG/OPERATOR

Sends a message to the system log or to the operator's console. OPERATOR can be specified only in the batch environment.

MESSAge message-options

Identifies message to be displayed.

MSG can be used in place of MESSAGE.

TEXT IS message-text

Specifies the text of a message to be displayed in an online map's message field or sent to a batch application and a system log file.

Message-text specifies either the name of a variable data field containing the message text or the text string itself, enclosed in single quotation marks.

The text string can contain up to 240 displayable characters.

CODE IS message-code

Specifies the message dictionary code of a message to be displayed in an online map's message field or sent to the log file in a batch application.

In a batch application, the message is also sent to the operator, if directed by the destination specified in the dictionary.

Message-code specifies either the name of a variable data field that contains the message code or the 6-digit code itself, expressed as a numeric literal.

PARMS = parameter

Specifies a replacement parameter for each variable field in the stored message identified by *message-code*.

Up to nine replacement parameters can be specified for a message. Multiple parameters must be specified in the order in which they are numbered and separated by blanks or commas.

Parameter specifies either the name of an EBCDIC or unsigned zoned decimal variable data field that contains the parameter value or the actual parameter value, enclosed in single quotation marks.

The parameter value must contain displayable characters. At run time, each variable data field in a stored message expands or contracts to accommodate the size of its replacement parameter. A replacement parameter can be a maximum of 240 bytes.

PREFIX IS prefix

Overrides the default prefix of a dialog and a map.

Prefix must either specify an EBCDIC or unsigned zoned decimal variable data field that contains a 2-character prefix or the 2-character prefix itself, enclosed in single quotation marks

Usage

Considerations

- Up to nine replacement parameters can be specified for a message.
- Multiple parameters must be separated by blanks or commas.
- Multiple parameters must be specified in the order in which they occur in the stored message.

Chapter 21. Cooperative Processing Commands

21.1 Using SEND/RECEIVE commands	21-3
21.1.1 How cooperative processing works	21-3
21.2 Sample cooperative application	21-4
21.2.1 Program A: Client listing (PC)	21-5
21.2.2 Dialog B: Server listing (Mainframe)	21-7
21.3 SEND/RECEIVE commands	21-9
21.4 ALLOCATE	21-10
21.5 CONFIRM	21-13
21.6 CONFIRMED	21-14
21.7 CONTROL SESSION	21-15
21.8 DEALLOCATE	21-17
21.9 PREPARE-TO-RECEIVE	21-19
21.10 RECEIVE-AND-WAIT	21-20
21.11 REQUEST-TO-SEND	21-21
21.12 SEND-DATA	21-22
21.13 SEND-ERROR	21-24
21.14 Design guidelines	21-25
21.15 Understanding conversation states	21-26
21.15.1 Conversation states in a successful data transfer	21-28
21.16 Testing APPC status codes and system fields	21-30
21.16.1 Status codes	21-30
21.16.2 System fields	21-30
21.16.3 When APPC status codes and system field values are returned	21-30
21.16.4 APPCCODE and APPCERC	21-31
21.16.5 System fields	21-34

21.1 Using SEND/RECEIVE commands

SEND/RECEIVE commands allow you to create applications that execute cooperatively on two systems. You can exchange information between a CA-ADS application (on the mainframe) and:

- A CA-ADS application (running under a different CA-IDMS/DC system)
- An Assembler program (running under CA-IDMS/DC)
- Any program using APPC/LU6.2, regardless of the program's platform

Applications that execute cooperatively on two systems are taking advantage of **cooperative processing**.

Client and server: With cooperative processing, labor is divided so that one side of the application acts as a **client** and the other acts as a **server**. The client provides front-end processing for the user (like data input, validation, and display). The server provides back-end processing (like database access and the implementation of business rules and procedures).

21.1.1 How cooperative processing works

The SEND/RECEIVE commands in CA-ADS follow the standards set for Advanced Program to Program Communication (APPC).

APPC is an IBM standard that provides enhanced Systems Network Architecture (SNA) support for distributed processing. APPC enables 2 processors to work together: it describes the protocols the 2 processors' programs use to communicate as they execute a single distributed transaction.

APPC is composed of logical and physical definitions of the system network. The logical component is the LU 6.2 protocol, which defines the rules that govern the exchange of information between the 2 programs.

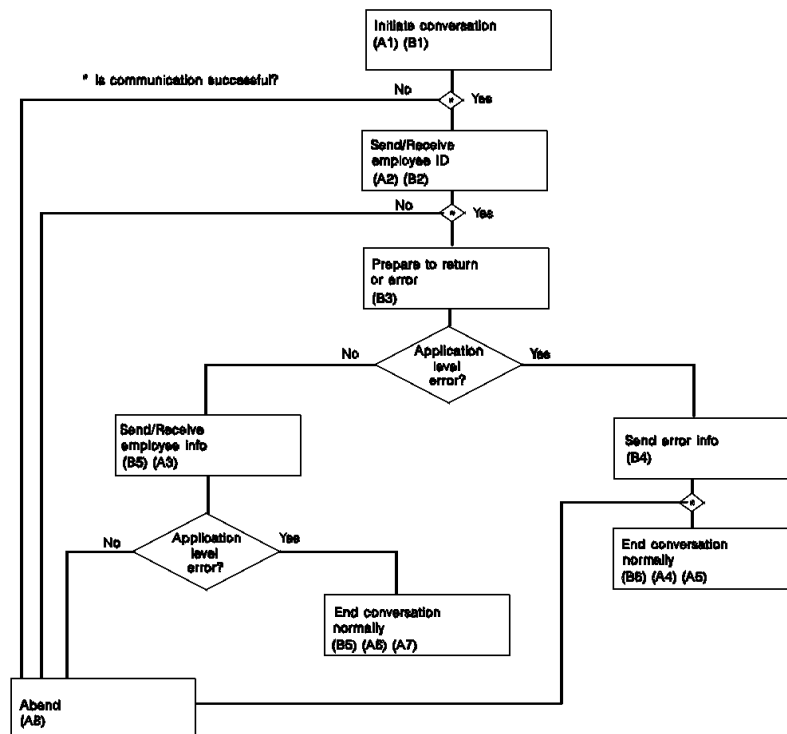
The LU 6.2 protocol structures a program-to-program conversation like a polite conversation: one side talks and the other side listens.

- One program (the **primary** program) starts the conversation by calling the other program (the **secondary** program).
- The 2 programs agree on the rules governing the conversation before the conversation can continue.
- The conversation goes back and forth, with the current speaker (in **send state**) always in control:
 1. When the listener (in **receive state**) wants to speak, the listener requests permission.
 2. If the speaker approves the request, the listener switches to send state and the speaker to receive state.

21.2 Sample cooperative application

This application retrieves employee information from the database. A user on the PC uses Program A (the client) to send an employee ID to the mainframe. Dialog B (the server) on the mainframe returns employee information to the PC.

The flowchart below describes the flow of information in the cooperative application. The communication commands in Program A and Dialog B are labeled in sequence (A1 through A8 and B1 through B6).



21.2.1 Program A: Client listing (PC)

```

!*****
!
! This module sends up the employee ID and receives employee
! data back from mainframe server dialog A240D1.

LOCAL Emp_data_group DICT.

SHOW TEXT '&nAccessing remote server ...'.

! Since we are going to use this data in some meaningful fashion we
! define a formatted conversation. Note the use of "FORMAT". This
! means that all data is automatically translated from PC data types
! to mainframe data types. IBM APPC LU 6.2 calls for the use of a return
! code to verify the state of a particular APPC verb. We use "Appccode".

(A1)      ALLOCATE CONNECT 'SYSTEM55' TPN 'A240D1' FORMAT.
          AFTER Comm-error CALL SR_Abend.

! Note the use of a local subroutine called SR_Abend.
!
! Let's send up the employee ID.
!

(A2)      SEND-DATA Emp_id.
          AFTER Comm-error CALL SR_Abend.

!
! "Turn the line around" and wait for server to send down
! the requested data. This flushes the communications
! buffer and passes control to the server dialog.

(A3)      RECEIVE-AND-WAIT Emp_data_group.
!
! If an employee was not found, then the server's SEND-ERROR
! shows PROG-ERROR at the client side of the conversation.
!

          AFTER Prog-error
          DO.
! One more receive to get the deallocate state...
(A4)      RECEIVE-AND-WAIT.
          AFTER Deallocate-normal
          DO.
(A5)      DEALLOCATE LOCAL.
          AFTER Comm-error CALL SR_Abend.
          INITIALIZE ( emp_first_name, emp_last_name,
                      office_code, emp_street, emp_city,
                      emp_state, emp_zip_first_five, status ).
          DISPLAY TEXT '&wEmployee not on file. Try again.'.
          END.
END.

```

```

                IF Appccode LT 0 OR What-received NE 'DATA-COMPLETE'
                    CALL SR_Abend.
!
! We seem to have received the data ok.
!
! The server is still in control of the conversation, so we
! do one more receive-and-wait to receive the fact that the
! server has deallocated. This puts the client in deallocate state,
! allowing the client to deallocate normally and regain control.
!
(A6)            RECEIVE-AND-WAIT.
                AFTER Deallocate-normal
                DO.
(A7)            DEALLOCATE LOCAL.
                ! Now move data to form fields and display it.
                Emp_first_name      = Emp_data_group.emp_first_name.
                Emp_last_name       = Emp_data_group.emp_last_name.
                Emp_street          = Emp_data_group.emp_street.
                Emp_city            = Emp_data_group.emp_city.
                Emp_state           = Emp_data_group.emp_state.
                Emp_zip_first_five  = Emp_data_group.emp_zip_first_five.
                Status              = Emp_data_group.status.
                Office_code         = Emp_data_group.office_code.

                DISPLAY FIELD Emp_id TEXT ' '.
                END.

                AFTER Comm-error CALL SR_Abend.

!***** Local subroutine *****

                DEFINE SR_Abend.

                SHOW TEXT '&cComm Error. APPCCODE='&svb.&svb. Appccode
                        &svb.&svb. ', APPCERC=' &svb.&svb. Appcerc.
(A8)            DEALLOCATE ABEND.
                DISPLAY.
!
```

21.2.2 Dialog B: Server listing (Mainframe)

```

!*****
DIALOG A240D1 (CA-ADS)

PROCESS NAME IS A240D1-PREMAP

! THE INTENT OF THIS MAPLESS DIALOG IS TO RECEIVE THE EMPLOYEE-ID
! FROM THE PC APPLICATION AND THEN OBTAIN THE APPROPRIATE RECORDS FROM
! THE CA-IDMS/DB DATABASE, BASED ON THE CONTENTS OF THE EMPLOYEE ID.
! THEN CERTAIN ELEMENTS IN THE RECORDS ARE SENT BACK TO THE PC FOR FURTHER
! PROCESSING.
!

! IN ALL APPC CONVERSATIONS THERE ARE LOCAL AND REMOTE PROGRAM.
! ONLY ONE OF THE PROGRAMS CAN CONTROL THE CONVERSATION BUT CONTROL
! CAN BE PASSED BACK AND FORTH BETWEEN LOCAL AND REMOTE APPLICATIONS.
!

! IN THIS APPLICATION THE DEFAULT SENDER IS THE PC PROGRAM BECAUSE IT
! ISSUED THE ALLOCATE. ON THE RECEIVING END A "CONTROL SESSION" IS
! INITIATED IN RESPONSE TO THE ALLOCATE.
!

! NOTE THAT THE USE OF THE FORMAT/NOFORMAT PARAMETERS MUST BE THE SAME AT
! BOTH ENDS OF THE CONVERSATION.

READY.
(B2)CONTROL SESSION FORMAT.
    IF APPCCODE LT 0 THEN
        CALL SR-ABEND.

! LET'S GET THE EMPLOYEE ID FROM THE PC.
(B2)RECEIVE-AND-WAIT EMP-ID-WORK.
    IF APPCCODE LT 0 OR WHAT-RECEIVED NE 'DATA-COMPLETE' THEN
        CALL SR-ABEND.

! ALL APPC COMMUNICATIONS OCCUR IN HALF-DUPLEX MODE. THAT IS, ONLY ONE
! OF THE PROGRAMS CAN TALK WHILE THE OTHER LISTENS. IN ORDER TO
! "TURN THE LINE AROUND" THE RECEIVER MUST WAIT UNTIL THE SENDER SAYS IT'S
! OK TO SEND. THIS IS ACCOMPLISHED BY WAITING FOR 'SEND' TO BE RECEIVED
! IN THE WHAT-RECEIVED SYSTEM VARIABLE.

```

```
(B3) RECEIVE-AND-WAIT.
    IF APPCCODE LT 0 OR WHAT-RECEIVED NE 'SEND' THEN
        CALL SR-ABEND.

    ! EMP-ID-0415 IS DEFINED AS PIC 9(4) USAGE IS DISPLAY. APPC PRESENTATION
    ! SERVICES DO NOT SUPPORT ZONED DECIMAL DATA TYPE. IN ORDER TO RECEIVE THE
    ! EMPLOYEE ID FROM THE PC, A WORK RECORD IS CREATED WITH USAGE IS COMP.
    ! THEN GET THE RECORD USING AN OBTAIN CALC.

    MOVE WK-EMP-ID TO EMP-ID-0415.
    OBTAIN CALC EMPLOYEE.

    IF DB-REC-NOT-FOUND THEN DO.
(B4)    SEND-ERROR.                ! NOTIFY CLIENT WE DIDN'T FIND EMPLOYEE
        IF APPCCODE LT 0 THEN CALL SR-ABEND.
(B5)    DEALLOCATE.
        LEAVE ADS.
    END.
    ! GET THE OFFICE RECORD, IF ONE EXISTS.

    IF SET OFFICE-EMPLOYEE MEMBER THEN
        OBTAIN OWNER WITHIN OFFICE-EMPLOYEE.

    MOVE EMP-FIRST-NAME-0415      TO WK-EMP-FIRST-NAME.
    MOVE EMP-LAST-NAME-0415      TO WK-EMP-LAST-NAME.
    MOVE EMP-STREET-0415         TO WK-EMP-STREET.
    MOVE EMP-CITY-0415           TO WK-EMP-CITY .
    MOVE EMP-STATE-0415          TO WK-EMP-STATE.
    MOVE EMP-ZIP-FIRST-FIVE-0415 TO WK-EMP-ZIP-FIRST-FIVE.
    MOVE STATUS-0415             TO WK-STATUS.
    MOVE OFFICE-CODE-0450        TO WK-OFFICE-CODE.

(B6) SEND-DATA WK-EMP-REC2.
    IF APPCCODE LT 0 THEN CALL SR-ABEND.

    ! NOW THAT WE HAVE FINISHED, LET'S FLUSH THE COMMUNICATIONS
    ! BUFFER AND TERMINATE THE CONVERSATION.

(B6) DEALLOCATE.
    LEAVE ADS.

!*****
!
! SEND/RECEIVE ERROR HANDLING SUBROUTINE
!
    DEFINE SR-ABEND.

(B7)    DEALLOCATE ABEND.
        LEAVE ADS.
```


21.3 SEND/RECEIVE commands

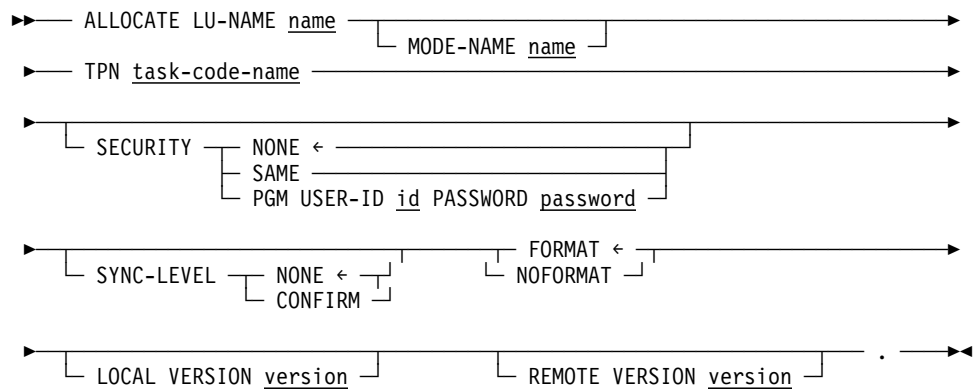
SEND/RECEIVE commands are listed below. Syntax and syntax rules for each command is presented in alphabetical order after the table.

Command	What it does	When it's issued
ALLOCATE	Begins a conversation with a server dialog	The first communication command issued by the client dialog when it's ready to communicate
CONFIRM	Sends a confirmation request to the remote program and waits for a reply	Issued by the dialog in send state
CONFIRMED	Sends a confirmation reply to the remote program	Issued by the dialog in confirm state
CONTROL SESSION	Acknowledges the conversation and agrees to the rules governing the conversation; a CA extension to APPC used in PC-to-mainframe conversations, but not required for mainframe-to-mainframe communication	Issued by the server dialog in response to the client dialog's ALLOCATE; parameters must match those on the ALLOCATE
DEALLOCATE	Ends the conversation	The last command in the conversation issued by either dialog
PREPARE-TO-RECEIVE	Changes the local side to receive state	Issued by the dialog in send state in response to REQUEST-TO-SEND
RECEIVE-AND-WAIT	Waits for a response from the remote program and receives the data or a value in the system field, WHAT-RECEIVED, upon arrival	Issued by a dialog that wants to receive data (the dialog can be in send or receive state); when used in send state, it indicates that the local dialog wants to receive data, allowing the remote dialog to send data
REQUEST-TO-SEND	Notifies the remote program that the local program is ready to send data	Issued by the dialog in receive state
SEND-DATA	Sends data to the remote program	Issued by the dialog in send state
SEND-ERROR	Notifies the other dialog of an application-level error	Issued by either dialog

21.4 ALLOCATE

Purpose: Begins a conversation between a CA-ADS dialog and a remote dialog or program. The FORMAT/NOFORMAT setting on the ALLOCATE command must match the FORMAT/NOFORMAT setting on the CONTROL SESSION command.

Syntax:



Parameters

LU-NAME name

Specifies a field or string that identifies the 1- through 8-character name of the logical unit used by the remote dialog.

Name must match the logical unit name of an APPC line defined to the local CA-IDMS/DC system.

MODE-NAME name

Specifies the name used by the remote logical unit to select the mode of transmission for the conversation.

Name Is either a 1- through 8-character mode name or a variable containing the mode name.

If omitted, CA-IDMS/DC uses the mode name defined to the APPC line.

TPN task-code-name

A variable or string that contains or specifies the name of the remote program to be initiated by the ALLOCATE command.

If trying to initiate a mainframe CA-ADS task, the *task-code-name* must be a 1- through 8-character task code defined to the remote CA-IDMS/DC system that invokes ADSORUN1.

SECURITY

Provides security information to the remote program.

NONE

Specifies that no security information is required for the conversation.

NONE is the default for SECURITY.

SAME

Specifies that the signon user ID is passed to the remote program. The following considerations apply:

- This signon does not work if a password is required to sign on to a *separate* CA-IDMS/DC or CICS system
- This password does not allow you to sign on to the *same* CA-IDMS/DC system (CA-IDMS/DC will not support 2 LTEs signed on for the same user at the same time)

PGM USER-ID id

Specifies the user ID of the user who runs the application.

Id is either a 1- through 32-character user ID or a field containing a user ID.

PASSWORD password

Introduces the password of the user who runs the application.

Password is either a 1- through 8-character password or a field containing a password.

This information is used to sign on to the remote logical unit.

SYNC-LEVEL

Introduces the level of synchronization to use for the conversation.

NONE

Specifies that no confirmation commands can be used.

CONFIRM

Specifies that confirmation commands can be used.

FORMAT

Specifies that data will be converted by APPC presentation services before the receiving program sees it:

- Text is converted between ASCII and EBCDIC.
- Numbers are converted between mainframe and PC format.

FORMAT is the default when neither FORMAT or NOFORMAT is specified.

NOFORMAT

Specifies that no data will be converted. If data conversion is required, you must code any data translation or conversion.

LOCAL VERSION version

Specifies either a 1- through 32-character local program version identifier or a field containing a version ID sent to the remote program.

REMOTE VERSION version

Specifies a variable of at least 32 characters to receive the version identifier sent by the remote program.

Example: In order to allocate a conversation with another CA-ADS dialog on a different CA-IDMS/DC system, code:

ALLOCATE LU-NAME 'S75LU1' TPN 'DLG1' SECURITY NONE
SYSNC-LEVEL NONE NOFORMAT.

S75LU1 is the logical unit name of an APPC line defined to the local CA-IDMS/DC system, and DLG1 is the task code that initiates an CA-ADS dialog on the remote CA-IDMS/DC system. Security and confirmation are not being used, and conversion is not needed between 2 mainframe applications.

21.5 CONFIRM

Purpose: Sends a confirmation request to a remote program and waits for a reply.

Syntax:

►—— CONFIRM — . —————►◄

Usage: CONFIRM sends all data in the communications buffers to the remote program.

Considerations

- Before CONFIRM can be issued, the conversation must have a synchronization level of CONFIRM: SYNC-LEVEL CONFIRM on the ALLOCATE command and the issuing dialog must be in the send state.
- **When using CONFIRM to synchronize processing between 2 programs**
 - If the remote program received all data sent, it returns CONFIRMED.
 - If the remote program can't process the data due to an application-level error, it returns SEND-ERROR.
- **To check the status of a conversation**, test the system fields REQUEST-TO-SEND-RECEIVED and WHAT-RECEIVED.

Example: The local program can issue a confirmation or send an error message:

```
SEND-DATA EMPLOYEE-RECORD.
IF APPCCODE EQ ZERO
  THEN
    CONFIRM.
ELSE
  DO.
    DEALLOCATE ABEND.
    ABORT MSG TEXT 'SEND-DATA ERROR'.
  END.
```

For the corresponding response from the remote program, see CONFIRMED.

21.6 CONFIRMED

Purpose: Sends a confirmation reply to the remote dialog.

Syntax:

►—— CONFIRM — . —————►◄

Usage: CONFIRMED sends all data in the communications buffers to the remote program.

Note: CONFIRMED is sent in response to CONFIRM only.

Considerations

- Before CONFIRMED can be issued, the conversation must have a synchronization level of CONFIRM: SYNC-LEVEL CONFIRM on the ALLOCATE command.
- **When using CONFIRM to synchronize processing between 2 programs**
 - If the local dialog received all data sent, it returns CONFIRMED.
 - If the local dialog can't process the data due to an application-level error, it returns SEND-ERROR.
- **To check the status of a conversation**

Test the system fields: APPCCODE and WHAT-RECEIVED.

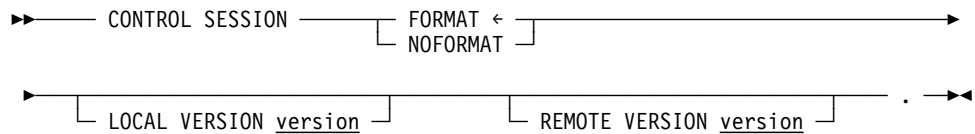
Example: In response to the preceding CONFIRM example, the remote program would issue:

```
RECEIVE-AND-WAIT EMPLOYEE-RECORD.  
IF APPCCODE EQ ZERO  
  THEN  
    DO.  
  RECEIVE-AND-WAIT.  
  IF APPCCODE EQ ZERO AND WHAT-RECEIVED EQ 'CONFIRM'  
    THEN  
      CONFIRMED.  
END.
```

21.7 CONTROL SESSION

Purpose: Issued by the secondary dialog in response to the ALLOCATE command sent by the primary dialog.

Syntax:



Parameters

FORMAT

Specifies that data will be converted by APPC presentation services before the receiving program sees it:

- Text is converted between ASCII and EBCDIC.
- Numbers are converted between mainframe and PC format.

FORMAT is the default for CONTROL SESSION.

NOFORMAT

Specifies that no data will be converted. If data conversion is required, you must code any data translation or conversion.

Note: The FORMAT/NOFORMAT setting on the ALLOCATE command must match the FORMAT/NOFORMAT setting on the CONTROL SESSION command.

LOCAL VERSION version

Specifies either a 1- to 32-character local program version identifier or a field containing a version ID sent to the remote program.

REMOTE VERSION version

Specifies a variable of at least 32 characters to receive the version identifier sent by the remote program.

Usage: CONTROL SESSION is issued in response to the ALLOCATE command sent by the primary program. These 2 commands establish the conventions governing the conversation.

This command is required in PC-to-mainframe conversations but is not used for mainframe-to-mainframe conversations.

Considerations: The FORMAT/NOFORMAT setting on the ALLOCATE command must match the FORMAT/NOFORMAT setting on the CONTROL SESSION command.

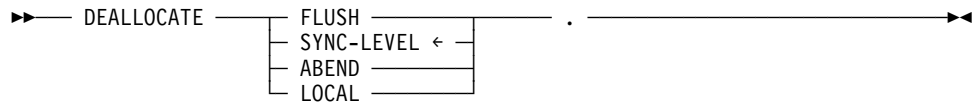
Example: In response to an ALLOCATE with the FORMAT setting, the local dialog sends LEVEL 1 in its local version to the remote dialog for testing. The remote dialog will receive it in the field, REMOTE-FIELD, for testing.

```
CONTROL SESSION FORMAT LOCAL VERSION 'LEVEL 1'
  REMOTE VERSION REMOTE-FIELD.
IF APPCCODE NE ZERO
  THEN
  DO.
  DEALLOCATE ABEND.
  ABORT MSG TEXT 'CONTROL SESSION ERROR'.
  END.
IF REMOTE-FIELD NE 'FIRST RELEASE'
  THEN
  DO.
  DEALLOCATE ABEND.
  ABORT MSG TEXT 'WRONG RELEASE OF PARTNER DIALOG'.
  END.
```


21.8 DEALLOCATE

Purpose: Ends the conversation.

Syntax:



Parameters

FLUSH

Sends all data in the communications buffer to the remote program and then terminates the conversation normally.

SYNC-LEVEL

Terminates the conversation based on the synchronization level. If the current synchronization level is:

- **NONE** — All data in the communications buffer is sent to the remote dialog and the conversation terminates normally.
- **CONFIRM** — All data in the communications buffer is sent to the remote dialog along with a request for confirmation. Once confirmation arrives from the remote program, the conversation terminates normally. Otherwise, the return code determines the state of the conversation.

SYNC-LEVEL is the default for DEALLOCATE

ABEND

Terminates the conversation abnormally.

The following considerations apply:

- If the dialog is in **send** state, all data in the buffer is sent to the remote program
- If the dialog is in **receive** state, all data in the buffer is purged

LOCAL

Terminates the conversation after the remote dialog has deallocated the conversation.

Usage: DEALLOCATE is the last command in the conversation, but each dialog can continue independent processing. The command only ends the conversation.

Considerations: DEALLOCATE:

- Is not a control command
- Does not end the dialog

Example

```
IF APPCCODE NE ZERO
  THEN
  DO.
  DEALLOCATE ABEND.
  ABORT MSG TEXT 'PROBLEM IN PROCESSING'.
  END.
```

21.9 PREPARE-TO-RECEIVE

Purpose: Prepares to receive data from a remote program.

Syntax:

```

➤➤— PREPARE-TO-RECEIVE — [ SYNC-LEVEL ←——— . —————➤➤
                             | FLUSH —————

```

Parameters

SYNC-LEVEL

Processes data based on the current synchronization level. If the synchronization level is:

- **NONE** — All data in the communications buffer is sent to the remote program before the local dialog is put in receive state.
- **CONFIRM** — All data in the communications buffer is sent to the remote program along with a request for confirmation. If the remote dialog issues **CONFIRMED**, the local dialog is put in receive state. Otherwise, the return code from the remote program determines the state of the conversation.

SYNC-LEVEL is the default for PREPARE-TO-RECEIVE

FLUSH

Sends all data in the communications buffer to the remote program.

Usage: PREPARE-TO-RECEIVE is issued by the dialog in send state in response to REQUEST-TO-SEND. PREPARE-TO-RECEIVE switches the local dialog to receive state.

Considerations: To check the status of a conversation test the system fields: REQUEST-TO-SEND and WHAT-RECEIVED.

21.10 RECEIVE-AND-WAIT

Purpose: Names the record or field to receive data.

Syntax:

►► — RECEIVE-AND-WAIT — variable-name — . — ◄◄

Parameters

variable-name

Names the record or field to receive data from the remote program.

Variable-name can be up to 32 characters long.

If you omit the *variable-name*, data received from the remote program is lost.
Only a value is received in the WHAT-RECEIVED field.

Usage:

Considerations: RECEIVE-AND-WAIT is issued by the dialog in receive state:

- **If presentation services are being used**, then the variable's definition must match the description of the incoming data definition.
- **If unformatted data is being received**, the maximum length received is derived from the variable's definition.

Example: This server dialog receives an employee id from the client and then obtains the appropriate employee records from the database. The first RECEIVE-AND-WAIT receives the id. The second turns the line around prior to sending information to the client.

```
CONTROL SESSION FORMAT.  
IF APPCCODE LT ZERO  
    THEN CALL SR-ABEND.  
RECEIVE-AND-WAIT EMP-ID-WORK.  
IF APPCCODE LT 0 OR WHAT-RECEIVED NE 'DATA-COMPLETE'  
    THEN CALL SR-ABEND.  
RECEIVE-AND-WAIT.  
.  
.  
.
```

21.11 REQUEST-TO-SEND

Purpose: Notifies the remote dialog that the CA-ADS application is ready to send data. The remote dialog can respond to this command in either of the following ways:

- Ignoring the request and continuing to send data
- Granting the request and starting to receive data

To determine whether permission to send has been granted, check the system field (WHAT-RECEIVED).

Syntax:

►►—— REQUEST-TO-SEND —— . ——►◄

21.12 SEND-DATA

Purpose: Sends data to the remote dialog.

Syntax:

►— SEND-DATA variable-name — . —————►◄

Parameters

variable-name

Specifies the name of the record or element to be sent.

Variable-name can be up to 32 characters long.

Note: The definition of *variable-name* must be in the dictionary associated with the local dialog.

Usage: Data to be sent is stored in the CA-ADS communications buffer. When the buffer is full or you issue a CONFIRM command or DEALLOCATE with the FLUSH option, the data in the buffer is sent to the remote dialog.

Considerations: CA-ADS issues a compiler error:

- If the record or field transmitted is not a valid data type (these data types are not valid for SEND/RECEIVE):
 - Pointer
 - Multibit binary
 - Zoned decimal
 - Graphics (kanji)
- If the record or field transmitted is coded:
 - With an OCCURS DEPENDING ON clause
 - As USAGE COMP with decimal positions
- If the record is a logical record
- If *variable-name* is a level 88
- If a group contains more than 256 fields
- If a record definition makes an FD longer than 32,768 bytes

Any field that redefines another field is ignored, but a transmitted field can be a REDEFINES field. For example, FIELD-B redefines FIELD-A within RECORD1. If you send RECORD1, FIELD-B is ignored and the record is sent as if FIELD-A is the current definition of the field.

Because only a subset of IDD data types are supported, you should:

1. Create a work record (much like the map work record) of the proper data types.

2. Move the values from your database record to this work record.
3. Move the received values back to your database record.

Example: After retrieving data and building the work record, send the data to the client dialog.

```
SEND-DATA WK-EMP-REC2.  
IF APPCCODE LT 0 THEN CALL SR-ABEND.
```

21.13 SEND-ERROR

Purpose: Informs the remote program that CA-ADS detected an application-level error (such as DB-REC-NOT-FOUND).

Syntax:

►—— SEND-ERROR —— . ——►

Usage: When you issue SEND-ERROR, the remote program does not automatically send the data again. The remote program must detect the SEND-ERROR and respond appropriately.

Example: If the record is not found, notify the client.

```
IF DB-REC-NOT-FOUND THEN DO.  
    SEND-ERROR.  
    IF APPCCODE LT 0 THEN CALL SR-ABEND.  
    DEALLOCATE.  
    LEAVE ADS.  
END.
```


21.14 Design guidelines

- **Be careful not to end a conversation inadvertently.**

A conversation is a task-level resource in CA-IDMS/DC. When the task ends, any ongoing conversation will be deallocated automatically. So do not use:

- DISPLAY
- LEAVE ADS NEXT TASK CODE
- LINK (to a program that will pseudoconverse)

Using any of these commands will deallocate your conversation.

- **Be aware of the location of the allocated dialog.**

The CA-ADS allocated task runs as a nonterminal task. Because you can not point to a secondary load area or load library on the ALLOCATE command, the allocated dialog should reside in the CA-IDMS/DC default load search sequence.

If you want to use a secondary load area or load library, you must override the search sequence by:

- Using the SECURITY parameter on the ALLOCATE command issued by the primary dialog.
- Dialogs which exist in secondary load areas can be accessed in the client task thread by using a signon profile associated with that user containing DCUF SET DICTNAME or LOADLIST to change the search sequence for the secondary dialog.

For server task threads a new system loadlist must be created and the secondary dictionary entry must be coded prior to the primary dictionary entry. A typical loadlist follows:

```
ADD LOADLIST NEWLOAD
  DICTNAME IS APPCDICT VERSION IS 1
  DICTNAME IS USER-DEFAULT VERSION IS USER-DEFAULT
  DICTNAME IS SYSTEM-DEFAULT VERSION IS SYSTEM-DEFAULT
  LOADLIB IS USER-DEFAULT
  DICTNAME IS USER-DEFAULT VERSION IS 1
  DICTNAME IS SYSTEM-DEFAULT VERSION IS 1
  LOADLIB IS SYSTEM-DEFAULT
MOD SYSTEM nnn
LOADLIST = NEWLOAD
```

21.15 Understanding conversation states

You must be aware of conversation states when you are developing a cooperative application. Whenever a dialog is involved in a conversation, it is in a specific conversation state. The state determines which communication operations can be performed at that time. For example, a dialog must be in send state to send data; and the partner dialog must be in receive state to receive data. You must keep both dialogs synchronized to allow information to be exchanged.

Valid conversation states

State	What it means
Reset	No conversation exists
Send	The dialog can send data, request confirmation, or deallocate the conversation
Receive	The dialog can receive information from its partner dialog.
Confirm	The dialog can reply to a confirmation request (there are three types of confirm state, based on the state of the dialog after a communication command is issued)
Deallocate	The dialog can deallocate the conversation

Confirmation: The use of confirmation is optional. The primary program can request confirmation at the beginning of the session. If confirmation is used, the sending program can issue a CONFIRM command to which the recipient must respond CONFIRMED or SEND-ERROR. You can acknowledge the receipt of data programmatically if you prefer.

Statements and conversation states: The following table summarizes, for each communication command, the states in which the command can be issued and the resulting state after the command is executed.

To issue this command	The dialog must be in this state	After this return code	The dialog is in this state
ALLOCATE	Reset	OK	Send
		Other	Reset
CONFIRM	Send	OK	Send
		PROG-ERROR	Receive
		Other	Deallocate
CONFIRMED	ConfirmR	OK	Receive
		Other	Deallocate

To issue this command	The dialog must be in this state	After this return code	The dialog is in this state
	ConfirmS	OK	Send
		Other	Deallocate
	ConfirmD	Any	Deallocate
CONTROL	Receive	OK	Receive
SESSION		Other	Deallocate
DEALLOCATE			
FLUSH	Send	Any	Reset
CONFIRM	Send	OK	Reset
		PROG-ERROR	Receive
		Other	Reset
ABEND	Any	Any	Reset
LOCAL	Deallocate	Any	Reset
PREPARE-TO-RECEIVE	Send	OK or PROG-ERROR	Receive
		Other	Deallocate
RECEIVE-AND-	Send or receive	OK, Data complete	Receive
WAIT		OK, Send	Send
		PROG-ERROR	Receive
		Other	Deallocate
		OK, CONFIRM	ConfirmR
		OK, CONFIRM-SEND	ConfirmS
		OK, CONFIRM-	
		DEALLOCATE	ConfirmD
REQUEST-TO-SEND	Receive	OK	Receive
		Other	Deallocate
SEND-DATA	Send	OK	Send
		PROG-ERROR	Receive
		Other	Deallocate

To issue this command	The dialog must be in this state	After this return code	The dialog is in this state
SEND-ERROR	Send or Receive	OK	Send
		PROG-ERROR	Receive
		Other	Deallocate
	ConfirmR	OK	Send
	ConfirmS	PROG-ERROR	N/A
	ConfirmD	Other	Deallocate

21.15.1 Conversation states in a successful data transfer

In the diagram below, for example, Dialog A on the PC establishes a conversation with Dialog B on the mainframe. Dialog A sends a request for employee information to Dialog B. Dialog B processes the request and returns a reply. Matching data is found and returned. This is the same application shown in the flowchart earlier in this chapter.

The state changes are noted under the communications commands. Refer back to the previous flowchart and the sample code if you wish. You can see that Dialog B uses a RECEIVE-AND-WAIT to switch the line (changing from receive to send state in preparation for returning data to the PC). Also note the state changes necessary to deallocate the conversation. (Here, WR represents the WHAT-RECEIVED system field.)

A successful transaction

Dialog A	Dialog B
(A1) ALLOCATE State (reset to send)	(B1) CONTROL SESSION State (receive)
(A2) SEND-DATA Emp-id State (send)	(B2) RECEIVE-AND-WAIT Emp-id State (receive) WR=DATA-COMPLETE
(A3) RECEIVE-AND-WAIT Emp-data State (send to receive)	(B3) RECEIVE-AND-WAIT State (receive to send) WR=SEND
(A6) RECEIVE-AND-WAIT. State (receive to deallocate)	(B5) SEND-DATA Emp-data State (send)
(A7) DEALLOCATE State (deallocate to reset)	(B6) DEALLOCATE State (send to reset)

In this transaction:

- (A1) initiates conversation with ALLOCATE.
- (B1) acknowledges conversation with CONTROL SESSION.
- (B2) prepares to receive a request with RECEIVE-AND-WAIT.
- (A2) issues SEND-DATA with the employee id as a parameter.
- (B3) prepares to send a reply with RECEIVE-AND-WAIT. This command switches the line. Dialog B changes state from receive to send.
- (A3) prepares to receive the employee data.
- (B5) returns employee data to Dialog A with SEND-DATA. Dialog A tests for the PROG-ERROR condition, but does not find it. Dialog A checks that the data is complete (APPCCODE=OK and WR=DATA-COMPLETE).
- (A6) once the data is successfully received, RECEIVE-AND-WAIT prepares to deallocate resources verifying that Dialog B (currently in control) is ready to end the conversation.
- (B6) flushes the communications buffer and terminates the conversation with DEALLOCATE.
- (A7) releases local resources with DEALLOCATE LOCAL.

21.16 Testing APPC status codes and system fields

You can test the values of these codes and fields in your dialogs to determine how information will be processed. For example, you can refer back to the sample code to see the use of APPCCODE in the server dialog.

21.16.1 Status codes

These codes report the status of each communications command performed:

- **APPCCODE** provides the category of the message returned from the communications services for the most recent command executed.
- **APPCERC** contains more detailed information about the message returned by APPCCODE.

21.16.2 System fields

These system fields track information received from the remote program:

- **WHAT-RECEIVED** tells you what was received from the remote program.
- **REQUEST-TO-SEND-RECEIVED** tells you whether or not the remote program is requesting to send data.

21.16.3 When APPC status codes and system field values are returned

These are the status codes and system fields returned by the communications commands.

Command	APPCCODE	APPCERC	RECEIVED	RECEIVED
ALLOCATE	•	•		
CONFIRM	•	•		•
CONFIRMED	•	•		
DEALLOCATE	•	•		
PREPARE-TO-RECEIVE	•	•		
RECEIVE-AND-WAIT	•	•	•	
REQUEST-TO-SEND	•	•		
SEND-DATA	•	•		•
SEND-ERROR	•	•		•

Keep in mind:

- Status codes are updated after each communications command executes.
- A condition can be reported when the communications command that caused the error executes or when a subsequent communications command executes.

21.16.4 APPCCODE and APPCERC

If an error description says *internal error*, request technical support from your site. If your technical support staff can't remedy the problem, make sure they have the APPCCODE and APPCERC before they call CA Technical Support.

0: OK: The communications command executed successfully. Control has returned to your CA-ADS application. The current state of the conversation depends on the specific communications command you issued.

APPCERC	What it means
0	Data is available for the dialog to receive
1	Information other than data is available for the dialog to receive

-1: Parameter check: There is a coding error in either the CA-ADS application or the remote dialog that must be corrected. The syntax is correct, but there is a mismatch of parameters passed between the two dialogs or the parameters supplied are invalid.

APPCERC	What it means
0	Internal error
10	You issued a CONFIRM command when the conversation was allocated with a synchronization level of NONE
30	Internal error
31	Internal error
32	Internal error
33	Internal error
34	Internal error
35	Internal error
36	Internal error

-3: Allocation error: The specified conversation cannot be allocated.

APPCERC	What it means
0	Internal error
1	The conversation cannot be allocated because of a condition that is not temporary (for example, a session protocol error). Do not retry the allocation request until the condition is corrected.
2	The conversation cannot be allocated because of a condition that can be temporary (for example, the secondary application is not available). If the condition is temporary, you can retry the allocation request.
3	The remote program rejected the allocation request because it did not understand the TPN. The TPN must be a task code associated with the dialog and defined to CA-IDMS/DC if you are trying to allocate an CA-ADS task.
4	The <i>task-code-name</i> specified on the ALLOCATE command exists but cannot be started. This is not a temporary condition and must be resolved by a systems programmer. Do not retry the ALLOCATE until the situation is corrected.
5	The <i>task-code-name</i> specified on the ALLOCATE command exists but cannot be started. This is a temporary condition. You can retry the allocation request.
6	The user specified on the SECURITY parameter of the ALLOCATE command is not known to the remote program.
7	Internal error.
8	Internal error.

-4: Resource failure: A resource failure terminated the conversation prematurely.

APPCERC	What it means
1	The resource failure is not temporary (for example, a session protocol error). Do not retry the transaction until the condition is corrected.
2	The resource failure can be temporary (for example, a power outage, a line failure, or a problem with a modem). You can retry the transaction.

-5: Deallocate condition: The remote program issued a DEALLOCATE command.

APPCERC	What it means
0	The deallocation was normal.
1	The remote program specified the ABEND option on the DEALLOCATE command or the remote program has abended. Any data remaining in the CA-ADS communications buffer is purged.

-6: Program error: The remote program issued the SEND-ERROR command. There is an error in the local application that must be corrected.

-7: SVC error: The remote program issued the SEND-ERROR command. There is an error in the local application that must be corrected.

-8: State error: There is a coding error in your CA-ADS application. The CA-ADS side of the conversation was not in the correct state to execute the communications command you specified: for example, you tried to issue SEND-DATA while in receive state.

In some cases, you can need to issue a DEALLOCATE ABEND to recover from this error.

-9: Unsuccessful: The conversation command was unsuccessful.

-10: Control session error: The CA-ADS side of the conversation issued an ALLOCATE command correctly. But the remote program did one of the following:

- Omitted the CONTROL SESSION command
- Transmitted a CONTROL SESSION command after another communications command
- Transmitted a CONTROL SESSION command whose parameters do not agree with the ALLOCATE command

This code is returned by the ALLOCATE command. This code can indicate an internal error.

-11: Format descriptor error: The CA-ADS side of the conversation received an internal error from presentation services about the format descriptors.

-12: Send-data error: The CA-ADS side of the conversation detected an internal error or a conversion error. This code is reported by the SEND DATA command.

-13: Receive format error: CA-ADS received an error in a formatted conversation. This indicates an internal error. This code is reported by the RECEIVE-AND-WAIT command.

21.16.5 System fields

WHAT-RECEIVED: This variable tells you what was received from the remote program. It is updated after the RECEIVE-AND-WAIT command is executed.

Contents	Meaning
DATA-COMPLETE	Data was received successfully.
CONFIRM	The remote dialog issued a CONFIRM command and expects the local dialog to reply with the CONFIRMED command.
CONFIRM-SEND	The remote dialog issued a PREPARE-TO-RECEIVE command with the CONFIRM option. The local dialog can reply with either a CONFIRMED or a SEND-ERROR command.
CONFIRM-DEALLOCATE	The remote dialog issued a DEALLOCATE command with the CONFIRM option. The local dialog can reply with either a CONFIRMED or a SEND-ERROR command.
SEND	The remote program is in receive state and the local dialog is now in send state. The local dialog can now issue a SEND-DATA command.

REQUEST-TO-SEND- RECEIVED: This variable tells you whether or not the remote dialog issued the REQUEST-TO-SEND command. This variable is updated after the CONFIRM or the SEND-DATA command executes.

Contents	Meaning
0	The remote program has not requested to send data
1	The remote program is requesting to send data

If the local dialog receives REQUEST-TO-SEND then REQUEST-TO-SEND-RECEIVED is set to 1.

The local dialog resets REQUEST-TO-SEND-RECEIVED to 0 after every CONFIRM, SEND-DATA, and SEND-ERROR command.

Chapter 22. OSCaR Commands

- 22.1 OSCaR command syntax 22-4
 - 22.1.1 OPEN 22-4
 - 22.1.2 SEND 22-5
 - 22.1.3 CLOSE 22-6
 - 22.1.4 RECEIVE 22-6
- 22.2 Sample OSCaR application 22-7
- 22.3 OSCaR to APPC Mapping 22-9

Purpose: OPEN, SEND, CLOSE, and RECEIVE (OSCaR) commands are an interface between mainframe CA-ADS dialogs. OSCaR commands run as APPC commands; that is, as LU6.2 between mainframes. If a mainframe application is accessing a remote data base rather than a remote application, DDS should be more efficient.

Concept: OSCaR commands are much simpler than the CA-IDMS SEND/RECEIVE verb set for APPC cooperative processing in that they provide only a subset of the complete APPC functionality and synchronization of conversation states is automatic. It is not necessary to understand the CA-IDMS SEND/RECEIVE verb set, the IBM APPC verb set, or Conversation States before using OSCaR commands. However, it is necessary to understand basic cooperative processing concepts.

►► For more information, see Chapter 21, “Cooperative Processing Commands.”

Coding considerations

- Only four commands are defined: OPEN, SEND, CLOSE, and RECEIVE
- APPC and OSCaR commands are mutually exclusive within a single dialog:
 - APPC verbs such as ALLOCATE, CONTROL SESSION, and SEND-DATA are not allowed in dialogs containing OSCaR verbs
 - The four OSCaR verbs are not allowed in dialogs containing APPC verbs
- APPC data areas WHAT-RECEIVED and REQUEST-TO-SEND-RECEIVED cannot be referenced in a dialog that contains OSCaR commands
- OSCaR has no parameters equivalent to the FORMAT or SYNC-LEVEL parameters on the APPC ALLOCATE command
- OSCaR verbs always run as NOFORMAT
- Confirmation of user-validated data content must be sent via a user-defined control record rather than as a separate CONFIRM or SEND-ERROR command
- Commands to perform synchronization of conversation states, (RECEIVE-AND-WAIT, PREPARE-TO-RECEIVE, and REQUEST-TO-SEND) are done automatically by the runtime system when needed. These commands are needed only when a command is issued and the line is in the wrong state.

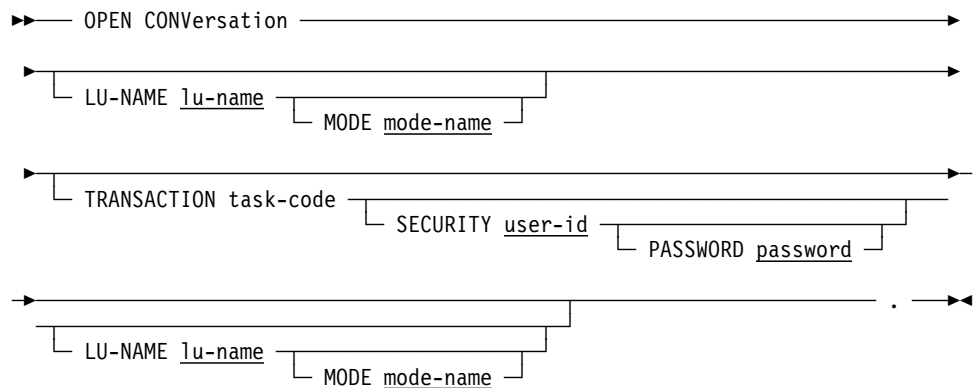
►► For more information, see 21.15, “Understanding conversation states.”
- Error conditions can be detected with autostatus or by examining ERROR-STATUS for 6901 or APPCCODE for a negative value

22.1 OSCaR command syntax

22.1.1 OPEN

Purpose: Establishes a conversation.

Syntax



Parameters

TRANSACTION task-code

Indicates that the dialog runs as a client; *task-code* names the task to be invoked on the remote logical unit.

If *task-code* names a mainframe CA-ADS dialog, it must be defined on the remote logical unit as invoking program ADSORUN1.

SECURITY user-id

Specifies the identifier of user to be signed on to the remote logical unit.

If SECURITY is not specified, signon is performed with no user identifier.

PASSWORD password

Specifies the password associated with *user-id* during signon to the remote logical unit. If PASSWORD is not specified, signon is performed with no password.

LU-NAME lu-name

Specifies a field or string that identifies the 1- through 8-character name of the logical unit used by the remote dialog.

Lu-name must match the logical unit name of an APPC line defined to the local CA-IDMS/DC system.

If LU-NAME is not specified, *lu-name* is the value of the ADSLUNAM attribute for the user session.

MODE mode-name

Specifies the name used by the remote logical unit to select the mode of transmission for the conversation.

Mode-name is either a 1- through 8-character mode name or a variable containing the mode name.

If MODE is not specified, *mode-name* is the value of the ADSMODE attribute for the user session. If there is no ADSMODE attribute, the value is the default mode name defined by the DLOGMOD parameter within the VTAM definition. If there is no default mode in the VTAM definition, the dialog aborts.

Usage

Placement of OPEN CONVERSATION: OPEN must be the first APPC command encountered in an OSCaR dialog.

OPEN CONVERSATION with no parameters: If no parameters are specified on OPEN CONVERSATION, the dialog runs as a server.

22.1.2 SEND

Purpose: Sends a data name to a remote logical unit.

Syntax

```

▶— SEND data-name —┬─ TRANSlate —┬─
                      └─ NOTRANSlate ┘

```

Parameters

data-name

Identifies the record or element name associated with the dialog to send to the remote logical unit.

TRANSlate

Not meaningful in a CA-ADS mainframe dialog.

NOTRANSlate

Not meaningful in a CA-ADS mainframe dialog.

Usage

Restrictions on data name: In a SEND command, *data-name* must not be or correspond to the name of:

- A logical record
- A built-in function
- An element defined as pointer, multi-bit binary, or graphic
- An 88-level element
- A reserved word, such as SPACES, DATE, CURSOR-ROW
- A quoted literal

22.1.3 CLOSE

Purpose: Ends a conversation.

Syntax

►► CLOSE CONvERsation _____ . ►►

Usage

Placement of CLOSE CONVERSATION: CLOSE CONVERSATION must be the last APPC command encountered in both client and server dialogs that use OSCaR commands.

22.1.4 RECEIVE

Purpose: Receives a data name from a remote logical unit.

Syntax

►► RECEIVE data-name

TRANslate
NOTTRANslate

 _____ . ►►

Parameters

data-name

Identifies the record or element name associated with the dialog to receive from the remote logical unit.

TRANslate

Not meaningful in a CA-ADS mainframe dialog.

NOTTRANslate

Not meaningful in a CA-ADS mainframe dialog.

Usage

Restrictions on data name: In a RECEIVE command, *data-name* must not be or correspond to the name of:

- A logical record
- A built-in function
- An element defined as pointer, multi-bit binary, or graphic
- An 88-level element
- A reserved word, such as SPACES, DATE, CURSOR-ROW
- A quoted literal

22.2 Sample OSCaR application

About this example: This sample application program retrieves EMPLOYEE/OFFICE data from a remote demo data base. It performs the same function as the example in 21.2, "Sample cooperative application." No intermediate records are needed because the OSCaR verbs support all data types found in the EMPLOYEE and OFFICE records.

Client map

RETRIEVE EMPLOYEE DATA

EMPLOYEE ID: ____

Employee name:
Office street:
Office city :

Enter any employee ID.
The employee's name and office address will be returned.

Client ENTER process

```

IF EMP-ID-0415 EQ ZEROES DO.                                !IF NO EMP-ID ENTERED
  INITIALIZE RECORDS (EMPLOYEE, OFFICE).                      !  CLEAR OLD DATA
  DISPLAY MESSAGE TEXT 'ENTER AN EMPLOYEE ID'.!              REQUEST EMP-ID
END.

IF FIELD EMP-ID-0415 IS CHANGED DO.                          !IF EMP-ID WAS ENTERED
  OPEN CONVERSATION TRANSACTION 'EMPSERVE'.
  SEND EMP-ID-0415.                                           !  SEND EMP-ID TO EMPSERV
  RECEIVE EMPLOYEE.                                           !  RETRIEVE EMPLOYEE
  RECEIVE OFFICE.                                             !  RETRIEVE OFFICE
  CLOSE CONVERSATION.
END.

IF EMP-NAME-0415 EQ ALL '*'                                  !DISPLAY RESULTS
  DISPLAY MESSAGE TEXT 'EMPLOYEE DOES NOT EXIST'.
ELSE
  DISPLAY MESSAGE TEXT 'EMPLOYEE DISPLAY IS COMPLETE'.

```

Server PREMAP process

```
!***** GET EMP-ID FROM DIALOG EMPCLIEN *****
OPEN CONVERSATION TRANSACTION 'EMPSEVE'.
RECEIVE EMP-ID-0415.

!***** GET EMPLOYEE/OFFICE DATA *****
OBTAIN CALC EMPLOYEE.
IF DB-STATUS-OK DO.
    IF SET OFFICE-EMPLOYEE MEMBER
        OBTAIN OWNER WITHIN OFFICE-EMPLOYEE.
    ELSE
        MOVE ALL '*' TO OFFICE-ADDRESS-0450.
    END.
ELSE DO.
    MOVE ALL '*' TO EMP-NAME-0415.
    !MIGHT INITIALIZE ALL EMPLOYEE FIELDS
    !EXCEPT EMP-NAME-0415 AND EMP-IF-0415.
    END.

!***** RETURN RECORDS TO CLIENT *****
SEND EMPLOYEE.
SEND OFFICE.
CLOSE CONVERSATION.
LEAVE ADS.
```

22.3 OSCaR to APPC Mapping

The following table outlines the conversions which can be used to map OSCaR commands to standard APPC commands. This will allow any CA-IDMS/ADS dialog using the OSCaR verb set to communicate with any other program using the standard LU6.2/APPC verb set.

Command	Present State	APPC Result	ADS Equivalent	#TREQ Assembler Equivalent
OPEN	Reset	ALLOCATE	Allocate...	#TREQ Alloc...
*With LU-NAME or ADSLUNAM defined				
LU-NAME and ADSLUNAM not defined	Reset	GET_ATTRIBUTES	Control-Session no format	#TREQ UIOCB...
SEND	Send	SEND_DATA	Send-Data	TREQ Put...
	Receive	REQUEST_TO_SEND	Request-To-Send	#TREQ Put optns= signal
		SEND_DATA	Send-Data	#Treq Put...
CLOSE	Deallocate	DEALLOCATE_LOCAL	Deallocate Local	#TREQ Get... (receive deallocate)
	Send	DEALLOCATE_SYNC_LEVEL	Deallocate Sync-Level	#TREQ Put optns= last
RECEIVE	Receive	RECEIVE_AND_WAIT	Receive-And-Wait	#TREQ Get...
	Send	PREPARE_TO_RECEIVE	Prepare-To-Receive	#TREQ Put optns= invite
		RECEIVE_AND_WAIT	Receive-And-Wait	#TREQ Get...

*ADSLUNAM is a user-defined User-Profile attribute. This can be used to define a default LU name for OPEN commands. Additionally, ADSMODE may be used to define a default MODE name for the OSCaR OPEN command. For more information on user profiles, see the *CA-IDMS Security Administration Guide*.

Appendix A. System Records

- A.1 Overview A-3
- A.2 ADSO-APPLICATION-GLOBAL-RECORD A-4
- A.3 ADSO-APPLICATION-MENU-RECORD A-15

A.1 Overview

CA-ADS provides the three system records, listed in the table below. This appendix describes ADSO-APPLICATION-GLOBAL-RECORD and ADSO-APPLICATION-MENU-RECORD.

►► For information about the system-supplied status definition record ADSO-STAT-DEF-REC, see Chapter 10, “Error Handling.”

CA-ADS System Records

System record	Purpose
ADSO-APPLICATION-GLOBAL-RECORD	Passes information between dialogs in an application
ADSO-APPLICATION-MENU-RECORD	Holds information for the runtime system to use in building menus.
ADSO-STAT-DEF-REC	Associates level-88 condition names with status codes during dialog compilation

ADSO-APPLICATION-GLOBAL-RECORD and ADSO-APPLICATION-MENU-RECORD have fully addressable fields. At runtime, information supplied during application definition is moved to the applicable fields in the system records.

When the fields of a system record are referenced by a dialog, the record must be associated with the dialog as a work or map record. In applications not defined using the application compiler, system records are treated like any other work or map records.

The CA-ADS system records ADSO-APPLICATION-GLOBAL-RECORD and ADSO-APPLICATION-MENU-RECORD are discussed separately below.

A.2 ADSO-APPLICATION-GLOBAL-RECORD

ADSO-APPLICATION-GLOBAL-RECORD is automatically associated with an application as a global record, unless the record is explicitly deselected on the Global Records screen while defining the application with the application compiler (ADSA). If ADSO-APPLICATION-GLOBAL-RECORD is deselected, on the Global Records screen, the runtime system does not supply runtime information to the application's dialogs, and dialogs cannot modify runtime flow of control by changing AGR-CURRENT-RESPONSE.

►► For information about global records, see Chapter 2, “CA-ADS Application Compiler (ADSA).”

For information about runtime flow of control, see Chapter 4, “CA-ADS Runtime System.”

The CA-ADS runtime system uses ADSO-APPLICATION- GLOBAL-RECORD in the following ways:

- To pass information about the current application to dialogs in the application
- To allow dialogs to modify the application flow of control (by modifying AGR-CURRENT-RESPONSE)
- To provide an additional means of passing information between dialogs (by assigning values to the AGR-PASSED-DATA and AGR-MESSAGE fields)

One copy of the record is automatically associated with all applications during application definition.

ADSO-APPLICATION-GLOBAL-RECORD is defined in the data dictionary as follows:

```

01  ADSO-APPLICATION-GLOBAL-RECORD.
03  AGR-APPLICATION-NAME    PICTURE IS X(8)    USAGE IS DISPLAY.
03  AGR-CURRENT-FUNCTION    PICTURE IS X(8)    USAGE IS DISPLAY.
03  AGR-NEXT-FUNCTION       PICTURE IS X(8)    USAGE IS DISPLAY.
03  AGR-CURRENT-RESPONSE    PICTURE IS X(8)    USAGE IS DISPLAY.
03  AGR-DEFAULT-RESPONSE    PICTURE IS X(8)    USAGE IS DISPLAY.
03  AGR-TASK-CODE           PICTURE IS X(8)    USAGE IS DISPLAY.
03  AGR-EXIT-DIALOG         PICTURE IS X(8)    USAGE IS DISPLAY.
03  AGR-PRINT-DESTINATION    PICTURE IS X(8)    USAGE IS DISPLAY.
03  AGR-DATE                PICTURE IS X(8)    USAGE IS DISPLAY.
03  AGR-USER-ID             PICTURE IS X(32)   USAGE IS DISPLAY.
03  AGR-SECURITY-CODE        PICTURE IS X(32)   USAGE IS DISPLAY.
03  AGR-INSTALLATION-CODE    PICTURE IS X(32)   USAGE IS DISPLAY.
03  AGR-PASSED-DATA          PICTURE IS X(32)   USAGE IS DISPLAY.
05  AGR-PASSED-ONE          PICTURE IS X(32)   USAGE IS DISPLAY.
05  AGR-PASSED-TWO          PICTURE IS X(32)   USAGE IS DISPLAY.
05  AGR-PASSED-THREE        PICTURE IS X(32)   USAGE IS DISPLAY.
05  AGR-PASSED-FOUR         PICTURE IS X(32)   USAGE IS DISPLAY.
03  AGR-APPLICATION-VERSION PICTURE IS S9(4)   USAGE IS COMP.
03  AGR-APPL-SECURITY-CLASS PICTURE IS S999   USAGE IS COMP.
03  AGR-RESP-SECURITY-CLASS PICTURE IS S999   USAGE IS COMP.
03  AGR-PRINT-CLASS         PICTURE IS S999   USAGE IS COMP.
03  AGR-MODE                PICTURE IS X(4)    USAGE IS DISPLAY.
88  AGR-STEP-MODE           USAGE IS CONDITION-NAME VALUE IS 'STEP'.
88  AGR-FAST-MODE           USAGE IS CONDITION-NAME VALUE IS 'FAST'.
03  AGR-DATE-FORMAT         PICTURE IS X      USAGE IS DISPLAY.
88  AGR-MMDDYY             USAGE IS CONDITION-NAME VALUE IS 'C'.
88  AGR-DDMMYY             USAGE IS CONDITION-NAME VALUE IS 'E'.
88  AGR-YYMMDD             USAGE IS CONDITION-NAME VALUE IS 'G'.
88  AGR-YYDDD              USAGE IS CONDITION-NAME VALUE IS 'J'.
03  AGR-AID-BYTE            PICTURE IS X      USAGE IS DISPLAY.
88  AGR-ENTER              USAGE IS CONDITION-NAME VALUE IS QUOTE.
88  AGR-PF1                USAGE IS CONDITION-NAME VALUE IS '1'.
88  AGR-PF2                USAGE IS CONDITION-NAME VALUE IS '2'.
88  AGR-PF3                USAGE IS CONDITION-NAME VALUE IS '3'.
88  AGR-PF4                USAGE IS CONDITION-NAME VALUE IS '4'.
88  AGR-PF5                USAGE IS CONDITION-NAME VALUE IS '5'.
88  AGR-PF6                USAGE IS CONDITION-NAME VALUE IS '6'.
88  AGR-PF7                USAGE IS CONDITION-NAME VALUE IS '7'.
88  AGR-PF8                USAGE IS CONDITION-NAME VALUE IS '8'.
88  AGR-PF9                USAGE IS CONDITION-NAME VALUE IS '9'.
88  AGR-PF10               USAGE IS CONDITION-NAME VALUE IS ':'.
88  AGR-PF11               USAGE IS CONDITION-NAME VALUE IS '#'.
88  AGR-PF12               USAGE IS CONDITION-NAME VALUE IS '@'.
88  AGR-PF13               USAGE IS CONDITION-NAME VALUE IS 'A'.
88  AGR-PF14               USAGE IS CONDITION-NAME VALUE IS 'B'.
88  AGR-PF15               USAGE IS CONDITION-NAME VALUE IS 'C'.
88  AGR-PF16               USAGE IS CONDITION-NAME VALUE IS 'D'.
88  AGR-PF17               USAGE IS CONDITION-NAME VALUE IS 'E'.
88  AGR-PF18               USAGE IS CONDITION-NAME VALUE IS 'F'.
88  AGR-PF19               USAGE IS CONDITION-NAME VALUE IS 'G'.

```

```

88 AGR-PF20      USAGE IS CONDITION-NAME VALUE IS 'H'.
88 AGR-PF21      USAGE IS CONDITION-NAME VALUE IS 'I'.
88 AGR-PF22      USAGE IS CONDITION-NAME VALUE IS '¢'.
88 AGR-PF23      USAGE IS CONDITION-NAME VALUE IS '.'.
88 AGR-PF24      USAGE IS CONDITION-NAME VALUE IS '<'.
88 AGR-PA1       USAGE IS CONDITION-NAME VALUE IS '%'.
88 AGR-PA2       USAGE IS CONDITION-NAME VALUE IS '>'.
88 AGR-PA3       USAGE IS CONDITION-NAME VALUE IS ','.
88 AGR-CLEAR     USAGE IS CONDITION-NAME VALUE IS ' '.
88 AGR-LPEN      USAGE IS CONDITION-NAME VALUE IS '='.
88 AGR-EOF       USAGE IS CONDITION-NAME VALUE IS 'N'.
88 AGR-IOERR     USAGE IS CONDITION-NAME VALUE IS 'O'.
88 AGR-SERR      USAGE IS CONDITION-NAME VALUE IS 'P'.
88 AGR-IERR      USAGE IS CONDITION-NAME VALUE IS 'R'.
88 AGR-OERR      USAGE IS CONDITION-NAME VALUE IS 'S'.
03 AGR-CURRENT-FUNC-TYPE PICTURE IS X      USAGE IS DISPLAY.
88 AGR-FUNCTION-DIALOG  USAGE IS CONDITION-NAME
                           VALUE IS 'D'.
88 AGR-FUNCTION-MENU    USAGE IS CONDITION-NAME
                           VALUE IS 'M'.
88 AGR-FUNCTION-SIGNON  USAGE IS CONDITION-NAME
                           VALUE IS 'S'.
03 AGR-NEXT-FUNC-TYPE    PICTURE IS X      USAGE IS DISPLAY.
88 AGR-NEXT-DIALOG      USAGE IS CONDITION-NAME VALUE IS 'D'.
88 AGR-NEXT-MENU        USAGE IS CONDITION-NAME VALUE IS 'G'.
88 AGR-NEXT-MENU-DIALOG USAGE IS CONDITION-NAME
                           VALUE IS 'M'.
88 AGR-NEXT-SIGNON      USAGE IS CONDITION-NAME VALUE IS 'N'.
88 AGR-NEXT-SIGNON-DIALOG USAGE IS CONDITION-NAME
                           VALUE IS 'S'.
88 AGR-NEXT-PROGRAM     USAGE IS CONDITION-NAME VALUE IS 'P'.
88 AGR-NEXT-SYSTEM-FUNC USAGE IS CONDITION-NAME
                           VALUE IS 'F'.
                           VALUE IS 'B'.
                           VALUE IS 'O'.
                           VALUE IS 'U'.
                           VALUE IS 'R'.
                           VALUE IS 'T'.
                           VALUE IS 'Q'.
88 AGR-NEXT-FORWARD     USAGE IS CONDITION-NAME VALUE IS 'F'.
88 AGR-NEXT-BACKWARD    USAGE IS CONDITION-NAME VALUE IS 'B'.
88 AGR-NEXT-POP         USAGE IS CONDITION-NAME VALUE IS 'O'.
88 AGR-NEXT-POPTOP      USAGE IS CONDITION-NAME VALUE IS 'U'.
88 AGR-NEXT-RETURN      USAGE IS CONDITION-NAME VALUE IS 'R'.
88 AGR-NEXT-TOP         USAGE IS CONDITION-NAME VALUE IS 'T'.
88 AGR-NEXT-QUIT        USAGE IS CONDITION-NAME VALUE IS 'Q'.
03 AGR-CTRL-COMMAND     PICTURE IS X      USAGE IS DISPLAY.
88 AGR-TRANSFER         USAGE IS CONDITION-NAME VALUE IS 'T'.
88 AGR-INVOKE           USAGE IS CONDITION-NAME VALUE IS 'I'.
88 AGR-LINK             USAGE IS CONDITION-NAME VALUE IS 'L'.
88 AGR-RETURN           USAGE IS CONDITION-NAME VALUE IS 'R'.

```

```

03  AGR-SIGNON-SWITCH      PICTURE IS X      USAGE IS DISPLAY
                             VALUE IS 'N'.
      88  AGR-SIGNON-NOT-DONE  USAGE IS CONDITION-NAME
                                   VALUE IS 'N'.
      88  AGR-SIGNON-OK      USAGE IS CONDITION-NAME VALUE IS 'Y'.
03  AGR-DIALOG-NAME        PICTURE IS X(8)    USAGE IS DISPLAY.
03  AGR-FUNC-DESCRIPTION   PICTURE IS X(28)    USAGE IS DISPLAY.
03  AGR-MESSAGE            PICTURE IS X(240)   USAGE IS DISPLAY.
03  AGR-SIGNON-REQMTS      PICTURE IS X      USAGE IS DISPLAY.
      88  AGR-SIGNON-REQUIRED  USAGE IS CONDITION-NAME
                                   VALUE IS 'R'.
      88  AGR-SIGNON-OPTIONAL  USAGE IS CONDITION-NAME
                                   VALUE IS 'O'.
      88  AGR-SIGNON-NOT-ALLOWED  USAGE IS CONDITION-NAME
                                   VALUE IS 'N'.
03  AGR-MAP-RESPONSE       PICTURE IS X(8)    USAGE IS DISPLAY.
03  FILLER                 PICTURE IS X(54)   USAGE IS DISPLAY.

```

Field descriptions

AGR-APPLICATION-NAME

Contains the name of the current application, as specified on the Main Menu during application definition.

This field is updated once at the beginning of the application.

AGR-CURRENT-FUNCTION

Contains the name of the current function.

This field is updated at the beginning of each function.

AGR-NEXT-FUNCTION

Contains the name of the next function to be executed. The next function is the function initiated by the response contained in the AGR-CURRENT-RESPONSE field.

This field is updated on mapin from the terminal.

If a process command modifies AGR-CURRENT-RESPONSE to change the flow of control, AGR-NEXT-FUNCTION does not have to be changed. On an EXECUTE NEXT FUNCTION command, the runtime system transfers control to the function associated with the response.

AGR-CURRENT-RESPONSE

Contains the name of the next response to be executed, as specified by the user.

This field is updated on each mapin from the terminal.

The runtime system executes the response in AGR-CURRENT-RESPONSE when it encounters an EXECUTE NEXT FUNCTION command. The value in AGR-CURRENT-RESPONSE can be overwritten by the premap or response process of a dialog function.

Note that if AGR-CURRENT-RESPONSE is modified by a process command, the runtime system does not perform security checking.

►► For more information on the AGR-CURRENT-RESPONSE field, see Chapter 4, “CA-ADS Runtime System.”

AGR-DEFAULT-RESPONSE

Contains the name of the default response (if any) for the current function, as specified on any Function Definition screen with ADSA during application definition.

This field is updated at the beginning of each function. If the function has no default response, AGR-DEFAULT-RESPONSE contains blanks.

AGR-TASK-CODE

Contains the task code entered by the user to initiate the application.

This field is updated once at the beginning of the application.

AGR-EXIT-DIALOG

Contains the name of the user exit dialog (if any) associated with the current function, as specified on the Function Definition screen during application definition.

This field is updated at the beginning of each function, and is blank if the function has no user exit dialog.

AGR-PRINT-DESTINATION

Contains the default print destination for the application, as specified on the Main Menu during application definition.

AGR-PRINT-DESTINATION can be specified in a WRITE PRINTER command to specify a print destination.

This field is updated once at the beginning of the application and is blank if the application has no default print destination.

AGR-DATE

Contains the current date in the format specified by the application developer on the Main Menu during application definition.

This field is updated at the beginning of each premap and response process, and each menu and menu/dialog function.

AGR-USER-ID

Contains the user id passed to ADSO-APPLICATION-GLOBAL- RECORD from the AMR-USER-ID field of ADSO- APPLICATION-MENU-RECORD.

This field is updated after a successful user signon to the application and is blank if signon is unsuccessful or is not performed.

AGR-SECURITY-CODE

Contains the security class associated with the user id, as returned by CA-IDMS/DC or CA-IDMS/UCF following successful execution of a system SIGNON function. AGR-SECURITY-CODE is treated as a 256-bit field, with each bit representing a security class, from 0 to 255. A bit is set to 1 if the user is authorized at that security class, and is set to 0 if the user is not.

AGR-INSTALLATION-CODE

Contains the installation-defined security code associated with the user id, as returned by CA-IDMS/DC or CA-IDMS/UCF following successful execution of a system SIGNON function.

AGR-PASSED-DATA

A group field that consists of the following elements:

AGR-PASSED-ONE

Contains data passed to ADSO-APPLICATION-GLOBAL-RECORD from the AMR-PASSING field of ADSO-APPLICATION-MENU-RECORD on mapin from the terminal.

Note that if the user does not enter data in AMR-PASSING, AGR-PASSED-ONE is not updated.

AGR-PASSED-TWO

A 32-byte field that the application developer can use as applicable.

AGR-PASSED-THREE

A 32-byte field that the application developer can use as applicable.

AGR-PASSED-FOUR

A 32-byte field that the application developer can use as applicable.

The runtime system never updates fields AGR-PASSED-TWO, AGR-PASSED-THREE, and AGR-PASSED-FOUR.

AGR-APPLICATION-VERSION

Contains the version number of the current application, as specified by the application developer on the Main Menu during application definition.

This field is updated once at the beginning of the application.

AGR-APPL-SECURITY-CLASS

Contains the security class associated with the current application, as specified during application definition.

This field is updated once at the beginning of the application.

AGR-RESP-SECURITY-CLASS

Contains the security class associated with the response contained in the AGR-CURRENT-RESPONSE field, as specified on the Response Definition screen during application definition.

This field is updated on mapin from the terminal.

Note that if AGR-CURRENT-RESPONSE is modified by a process command, the runtime system does not perform security checking.

AGR-PRINT-CLASS

Contains the default print class for the application, as specified on the General Options screen during application definition.

AGR-PRINT-CLASS can be specified in a WRITE PRINTER command to specify a print class.

This field is updated once at the beginning of the application and is blank if the application has no default print class.

AGR-MODE

Contains the value passed to ADSO-APPLICATION-GLOBAL- RECORD from the AMR-MODE field of ADSO-APPLICATION- MENU-RECORD. The following level-88 condition names are defined for AGR-MODE:

AGR-STEP-MODE

AGR-MODE contains the value STEP.

AGR-FAST-MODE

AGR-MODE contains the value FAST.

This field is updated at the beginning of the application with the default mode specified on the Main Menu during application definition.

AGR-DATE-FORMAT

Contains a value indicating the date format specified by the application developer on the Main Menu during application definition. The following level-88 condition names are defined for AGR-DATE-FORMAT:

AGR-MMDDYY

AGR-DATE-FORMAT contains the value C.

AGR-DDMMYY

AGR-DATE-FORMAT contains the value E.

AGR-YYMMDD

AGR-DATE-FORMAT contains the value G.

AGR-YYDDD

AGR-DATE-FORMAT contains the value J.

This field is updated once at the beginning of the application.

AGR-AID-BYTE

Contains the AID byte that represents the control key pressed by the user.

A level-88 condition name is defined for each possible value.

AGR-CURRENT-FUNC-TYPE

Contains a value indicating the type of function named in the AGR-CURRENT-FUNCTION field. The following level-88 condition names are defined for AGR-CURRENT-FUNC-TYPE:

AGR-FUNCTION-DIALOG

AGR-CURRENT-FUNC-TYPE contains the value D.

AGR-FUNCTION-MENU

AGR-CURRENT-FUNC-TYPE contains the value M.

AGR-FUNCTION-SIGNON

AGR-CURRENT-FUNC-TYPE contains the value S.

This field is updated at the beginning of each function.

AGR-NEXT-FUNC-TYPE

Contains a value indicating the type of the function named in the AGR-NEXT-FUNCTION field. The following level-88 condition names are defined for AGR-NEXT-FUNC-TYPE:

AGR-NEXT-DIALOG

AGR-NEXT-FUNC-TYPE contains the value D.

AGR-NEXT-MENU

AGR-NEXT-FUNC-TYPE contains the value G.

AGR-NEXT-MENU-DIALOG

AGR-NEXT-FUNC-TYPE contains the value M.

AGR-NEXT-SIGNON

AGR-NEXT-FUNC-TYPE contains the value N.

AGR-NEXT-SIGNON-DIALOG

AGR-NEXT-FUNC-TYPE contains the value S.

AGR-NEXT-PROGRAM

AGR-NEXT-FUNC-TYPE contains the value P.

AGR-NEXT-SYSTEM-FUNC

AGR-NEXT-FUNC-TYPE contains the value F, B, O, U, R, T, or Q.

AGR-NEXT-FORWARD

AGR-NEXT-FUNC-TYPE contains the value F.

AGR-NEXT-BACKWARD

AGR-NEXT-FUNC-TYPE contains the value B.

AGR-NEXT-POP

AGR-NEXT-FUNC-TYPE contains the value O.

AGR-NEXT-POPTOP

AGR-NEXT-FUNC-TYPE contains the value U.

AGR-NEXT-RETURN

AGR-NEXT-FUNC-TYPE contains the value R.

AGR-NEXT-TOP

AGR-NEXT-FUNC-TYPE contains the value T.

AGR-NEXT-QUIT

AGR-NEXT-FUNC-TYPE contains the value Q.

This field is updated on mapin from the terminal.

AGR-CTRL-COMMAND

Contains a value indicating the control command associated with the response named in the AGR-CURRENT-RESPONSE field, as specified on the Response Definition screen during application definition. The following level-88 condition names are defined for AGR-CTRL-COMMAND:

AGR-TRANSFER

AGR-CTRL-COMMAND contains the value T.

AGR-INVOKE

AGR-CTRL-COMMAND contains the value I.

AGR-LINK

AGR-CTRL-COMMAND contains the value L.

AGR-RETURN

AGR-CTRL-COMMAND contains the value R.

This field is updated on mapin from the terminal. If a process command modifies AGR-CURRENT-RESPONSE to change the flow of control, AGR-CTRL-COMMAND does not have to be changed. On an EXECUTE NEXT FUNCTION command, the runtime system uses the control command associated with the response.

AGR-SIGNON-SWITCH

Contains a value indicating whether a system SIGNON function was performed for the current application. The following level-88 condition names are defined for AGR-SIGNON-SWITCH:

AGR-SIGNON-NOT-DONE

AGR-SIGNON-SWITCH contains the value N.

AGR-SIGNON-OK

AGR-SIGNON-SWITCH contains the value Y.

AGR-DIALOG-NAME

Contains the name of the dialog or user program (if any) associated with the function named in the AGR-CURRENT-FUNCTION field.

This field is updated at the beginning of a dialog, menu/dialog, or user program function, and is blank if the function is not associated with a dialog or user program.

AGR-FUNC-DESCRIPTION

Contains the description of the function named in the AGR-CURRENT-FUNCTION field, as specified on the Function Definition screen during application definition.

This field is updated at the beginning of each function and is blank if the function does not contain a description.

AGR-MESSAGE

AGR-MESSAGE is a 240-byte field that the application developer can use, as necessary. The runtime system never updates this field. (This field can be used for pass data.)

AGR-SIGNON-REQMTS

Contains a value indicating the signon requirements for the current application, as specified during application definition. The following level-88 condition names are defined for AGR-SIGNON-REQMTS:

AGR-SIGNON-REQUIRED

AGR-SIGNON-REQMTS contains the value R.

AGR-SIGNON-OPTIONAL

AGR-SIGNON-REQMTS contains the value O.

AGR-SIGNON-NOT-ALLOWED

AGR-SIGNON-REQMTS contains the value N.

This field is updated once at the beginning of the application.

AGR-MAP-RESPONSE

Contains a response name entered by the user in a field that maps to AGR-MAP-RESPONSE. AGR-MAP-RESPONSE performs the same function as a \$RESPONSE map field. The application developer can initialize the AGR-MAP-RESPONSE field with a default response name.

If both AGR-MAP-RESPONSE and \$RESPONSE are defined for a map, a value in the AGR-MAP-RESPONSE field has precedence over a value entered in the \$RESPONSE field. Once the AGR-MAP-RESPONSE field is initialized with a value, that value remains in the AGR-MAP-RESPONSE field until it is reinitialized with a new value either by process code or by the user.

Note: The \$RESPONSE map field can also be initialized by process code, through the \$RESPONSE system-supplied data field.

For information about the \$RESPONSE system-supplied data field, see Chapter 11, “Variable Data Fields” or refer to *CA-IDMS Mapping Facility*.

►► For descriptions of the screens used during an application definition session using the CA-ADS application compiler, see Chapter 2, “CA-ADS Application Compiler (ADSA).”

Usage: The following example illustrates the role of ADSO-APPLICATION-GLOBAL-RECORD during runtime execution of an application.

During execution of a nonmenu dialog function, the user selects a response that initiates either the FORWARD or BACKWARD system function. The following values are established in ADSO-APPLICATION-GLOBAL-RECORD:

- AGR-NEXT-FUNCTION contains FORWARD or BACKWARD, as applicable.
- AGR-CURRENT-RESPONSE contains the name of the response that initiates the FORWARD or BACKWARD system function.
- AGR-AID-BYTE contains the AID byte representing the control key pressed by the user.
- AGR-NEXT-FUNC-TYPE contains F or B, as applicable.
- AGR-CTRL-COMMAND contains a blank (X'40').

All other fields in the record remain unchanged.

The dialog can now access these fields to do its own paging, provided that ADSO-APPLICATION-GLOBAL-RECORD is defined to the dialog as a work record. (Automatic paging by the runtime system is performed only for menu functions.)

►► For more information on the use of ADSO-APPLICATION-GLOBAL-RECORD, refer to *CA-ADS Application Design Guide*.

A.3 ADSO-APPLICATION-MENU-RECORD

The CA-ADS runtime system builds menus by storing information in ADSO-APPLICATION-MENU-RECORD. This record is associated with maps used by menu and menu/dialog functions.

The menu map can be system-defined or user-defined. If a menu map is user-defined, ADSO-APPLICATION-MENU-RECORD must be explicitly associated with the map when it is defined.

►► For more information on menu maps, see Chapter 4, “CA-ADS Runtime System.”

Usage: In a menu/dialog function, ADSO-APPLICATION-MENU-RECORD is initialized at the beginning of the dialog, and its fields are primed by the runtime system when the map is displayed.

Thus, for example, at the beginning of a menu/dialog, AMR-PASSING does not contain any value passed by the previous menu function (AGR-PASSED-ONE of ADSO-APPLICATION-GLOBAL-RECORD does); and any value moved to a field in ADSO-APPLICATION-MENU-RECORD will be overwritten when the menu map is displayed.

ADSO-APPLICATION-MENU-RECORD is defined in the data dictionary as follows:

```
01  ADSO-APPLICATION-MENU-RECORD.
03  AMR-PAGE                PICTURE IS S99      USAGE IS COMP.
03  AMR-TOTAL-PAGES        PICTURE IS S99      USAGE IS COMP.
03  AMR-NEXT-PAGE          PICTURE IS S99      USAGE IS COMP.
03  AMR-HEADING             PICTURE IS X(237)   USAGE IS DISPLAY.
03  AMR-HDG REDEFINES AMR-HEADING
05  AMR-HL1                PICTURE IS X(79)    USAGE IS DISPLAY.
05  AMR-HL2                PICTURE IS X(79)    USAGE IS DISPLAY.
05  AMR-HL3                PICTURE IS X(79)    USAGE IS DISPLAY.
03  AMR-DATE               PICTURE IS X(8)     USAGE IS DISPLAY.
03  AMR-DIALOG             PICTURE IS X(8)     USAGE IS DISPLAY.
03  AMR-RESPONSE-FIELD     PICTURE IS X(8)     USAGE IS DISPLAY.
03  AMR-MODE               PICTURE IS X(4)     USAGE IS DISPLAY.
03  AMR-PASSING            PICTURE IS X(32)    USAGE IS DISPLAY.
03  AMR-USER-ID            PICTURE IS X(32)    USAGE IS DISPLAY.
03  AMR-PASSWORD           PICTURE IS X(8)     USAGE IS DISPLAY.
03  AMR-SELECT-SECTION     USAGE IS DISPLAY    OCCURS 50 TIMES.
05  AMR-SELECT             PICTURE IS X        USAGE IS DISPLAY.
05  AMR-RESPONSE           PICTURE IS X(8)    USAGE IS DISPLAY.
05  AMR-KEY                PICTURE IS X        USAGE IS DISPLAY.
05  AMR-DESCRIPTION        PICTURE IS X(28)    USAGE IS DISPLAY.
```

Field descriptions

AMR-PAGE

Maps to the PAGE field.

AMR-PAGE contains the number of the currently displayed page of the menu screen.

AMR-TOTAL-PAGES

Maps to the OF field.

AMR-TOTAL-PAGES contains the total number of pages for the current menu.

AMR-NEXT-PAGE

Maps to the NEXT PAGE field.

AMR-NEXT-PAGE contains the number of the next page to be displayed, as entered by the user.

AMR-HEADING

(or AMR-HDG) Maps to the heading text area.

AMR-HEADING or AMR-HDG contains the heading text specified by the application developer on the Function Definition (Menu) screen during application definition.

AMR-DATE

Maps to the DATE field.

AMR-DATE contains the current date in the format specified by the application developer on the General Options screen during application definition.

AMR-DIALOG

Maps to the DIALOG field.

If the current menu is associated with a dialog, AMR-DIALOG contains the name of the menu/dialog, as specified on the Response/Function List screen during application definition.

AMR-RESPONSE-FIELD

Maps to the RESPONSE field.

AMR-RESPONSE-FIELD contains the name of the next response to be executed, as entered by the user.

AMR-RESPONSE-FIELD is initialized with the default response for the current function (if any), as specified on the Function Definition screen during application definition. If the user does not specify a response, the default response remains in AMR-RESPONSE-FIELD.

AMR-MODE

Maps to the MODE field.

MODE contains the execution mode for the function, as specified on the Main Menu during application definition. At runtime, the user can change the specification in the MODE field on a menu map, thereby modifying the value in AMR-MODE. The value in AMR-MODE is passed to the AGR-MODE field of ADSO- APPLICATION-GLOBAL-RECORD each time the menu is mapped in.

AMR-PASSING

Maps to the SEND DATA field.

AMR-PASSING contains data to be passed to the next function, as entered by the user. CA-ADS transfers the contents of AMR-PASSING to the AGR-PASSED-ONE field of ADSO- APPLICATION-GLOBAL-RECORD. The runtime system reinitializes the AMR-PASSING field each time the menu is mapped out.

AMR-USER-ID

Maps to the ENTER USER ID field of a signon menu.

AMR-USER-ID contains the user id entered by the user.

CA-ADS transfers the contents of AMR-USER-ID to the AGR- USER-ID field of ADSO-APPLICATION-GLOBAL-RECORD. If a system SIGNON function is initiated, the runtime system passes the value in AMR-USER-ID to CA-IDMS/DC or CA-IDMS/UCF (DC/UCF) for security verification.

AMR-PASSWORD

Maps to the PASSWORD field of a signon menu.

AMR-PASSWORD contains the password entered by the user.

If a system SIGNON function is initiated, the runtime system passes the value in AMR-PASSWORD to DC/UCF for security verification. After passing the value, CA-ADS overwrites the AMR-PASSWORD field with blanks.

AMR-SELECT-SECTION

Maps to the response listing area.

AMR-SELECT-SECTION is a group field that occurs 50 times.

Each valid response associated with the current function is moved to an occurrence of AMR-SELECT-SECTION. The occurrences are mapped out to the menu screen one page at a time.

Each occurrence of AMR-SELECT-SECTION consists of the following elements:

AMR-SELECT

Contains the character entered by the user in the one-byte field provided to select the response.

AMR-RESPONSE

Contains the name of the response, as specified on the Response/Function List screen during application definition.

AMR-KEY

Contains the AID byte representing the control key that initiates the response, as specified on the Response/Function List screen during application definition.

AMR-KEY is translated by a code table to a five-byte map field in order to display the associated control key.

AMR-DESCRIPTION

Contains the description of the response, as specified on the Response Definition screen during application definition.

►► For descriptions of the screens used during an application definition session using the CA-ADS application compiler (ADSA), see Chapter 2, “CA-ADS Application Compiler (ADSA).”

Appendix B. CA-ADS Dialog and Application Reporter

- B.1 Overview B-3
- B.2 Dialog reports B-4
- B.3 Application reports B-15
- B.4 Control statements B-16
 - B.4.1 APPLICATIONS B-16
 - B.4.2 DIALOGS B-18
 - B.4.3 LIST B-21
 - B.4.4 SEARCH B-22
- B.5 SYSIDMS parameter file B-24
- B.6 JCL and commands to run reports B-25

B.1 Overview

The CA-ADS dialog and application reporter (ADSORPTS) is used to request batch reports about dialogs and applications. Reports can be summary or detailed. One dialog and/or application can be reported on, or several. Dialogs and applications to be included can be specified as a list of names, name ranges, and mask values.

Additional reports (AREPORTs) also provide information about dialogs and their components that are stored in the data dictionary. The information provided by each report is shown below.

►► For more information about these reports, see *CA-IDMS Reports*.

AREPORTs documenting CA-ADS dialogs

Report	Description
1	Lists detail information about dialogs and their components
2	Lists information about specified dialogs
3	Lists all dialogs associated with specified processes
4	Lists all dialogs associated with specified records.
5	Lists all dialogs associated with specified subschemas
6	Lists all dialogs associated with specified maps

This appendix provides the following information about ADSORPTS:

- A description of the dialog reporting capabilities
- A description of the application reporting capabilities
- Syntax rules for control statements
- JCL and commands for running reports

B.2 Dialog reports

The dialog reporting capabilities of ADSORPTS enable the application developer to request any or all of the following reports for one or several dialogs:

Summary reports: These list the following information about the object dialog:

- The name and version number of the dialog, and the date and time at which the dialog was compiled
- The name and version number of the map associated with the dialog, and the date and time at which the map was compiled
- The name and version number of the schema associated with the dialog
- The name of the subschema associated with the dialog
- The dialog's autostatus specification
- The dialog's FDB size

Processes reports: These list the module source statements for the premap and response processes associated with the object dialog. Source statements from included process modules are listed separately.

For each module listed, the following information is provided:

- The name and version number of the module
- The date on which the module was created, the date on which the module was last modified, and the ids of the users who created and modified the module

Note: The cross-reference report options are available with processes.

Records reports: These list the following information:

- The dictionary definitions for all records associated with the dialog
- The decimal position and hexadecimal offset of the fields in the listed records
- The lengths of the fields in the listed records

FDBLIST reports: These list the contents of the Fixed Dialog Block (FDB) for the dialog. The FDBLIST report includes the following information:

- General information, such as the dialog's name and compilation date, its map name and compile date, and its subschema description.
- Record descriptions contained in the Record Description Elements (RDEs).
- Premap process information contained in the Premap Process Element (PME).
- Response process information contained in the Response Process Elements (RSEs).
- Object code contained in the Process Object Code Table. One Process Object Code Table is associated with each PME and RSE of an FDB that contains object code.

- Command information contained in the Command Element (CME). One or more CMEs can be associated with a PME or RSE.
- Executable code and vector call information contained in the Executable Code/Vector Call Offset Table. One Executable Code/Vector Call Offset Table is associated with each PME and RSE of an FDB that contains object code.
- Included process module information in the Included Module Table (MDTA). One MDTA is displayed for each process in a dialog that has included modules.

Note that the representation of premap and response process information in a dialog's FDB and, consequently, the representation of this information in the report, depend on how the dialog was compiled, as follows:

- If the dialog was compiled with the symbol table option enabled, premap and response process commands are converted in the FDB to a series of command elements (CMEs).
- If the dialog was compiled without the symbol table option, premap and response process commands are converted in the FDB to object code. The object code for a command can be either an item of executable code or a vector call that references a CME.

►► For more information on using the symbol table option, see Chapter 3, “CA-ADS Dialog Compiler (ADSC).”

Note: Dialogs being reported on by ADSORPTS should be compiled with the diagnostic tables option enabled. ADSORPTS uses diagnostic tables to format premap and response process information. If a dialog's FDB does not contain diagnostic tables, ADSORPTS produces an unformatted report wherever formatting is not possible.

For more information on compiling a dialog with diagnostic tables, see Chapter 3, “CA-ADS Dialog Compiler (ADSC).”

Fixed Dialog Block field descriptions: The following table lists the fields displayed in the FDBLIST report.

Group	Field	Description
FDB	ID	Fixed dialog block identifier
	NAME	Dialog name
	DATE	Date dialog compiled
	TIME	Time dialog compiled
	MPNM	Map name
	MPDT	Date map compiled
	MPTM	Time map compiled

Group	Field	Description
	SCHNM	Schema name
	SSNM	Subschema name
	RDEA	Offset — start of record table
	PMEA	Offset — start of premap element
	RSEA	Offset — start of response table
	LITA	Offset — start of literal pool
	SSANA	Offset — subschema area name table
	NSSAN	Number of subschema area names
	SVER	Schema version
	MPVER	Map version
	DVER	Dialog version
	NRECS	Number of map records
	NFLDS	Number of map fields
	NDREC	Number of dialog records
	RSPMI	MRE index of map response field
	MSGMI	MRE index of map message field
	SEGVW	MRB — subschema segmented view
	FLAG	Fixed dialog block flag byte
	LREA	Offset — first logical record RDE
	ASRA	Offset — status definition record ASR
	RLSE	CA-ADS release
	FLAG2	FDB flag byte 2
	MAPPG	Map paging type
	HEXTA	Offset — FDB header extension area
	MDBO	Offset — map descriptor block (MDB)
	FLAG3	FDB flag byte 3
	PREFX	Message prefix
	DRSPO	Offset — default response process
	FDEO	Offset — format description headers (FDH) and elements (FDE)

Group	Field	Description
FHE (FDB header extension)	NODE	Alternate DB name
	DICT	Alternate dictionary name
	SDDN	Suspense file DD name
	DCLA Offset — SQL declaration process	
	SQLAM	SQL Access module name
	SQLTM	SQL time format
	SQLDT	SQL date format
	SQLFL	SQL compliance flag
MDB (map descriptor block)	MPNAM	Map name (batch)
	NEXT	Offset — next MDB
	DATE	Date map compiled
	TIME	Time map compiled
	VER	Version
	NRECS	Number of records
	NFLDS	Number of fields
	RSPMI	MRE index of map response field
	FLG1	Flag byte 1
	DDNAM	File/ddname
	CRECL	Compressed length for output map external record
	RECL	Real external record length
	CRECO	Offset — compressed external record
SSAN		Subschema area names
ASR	NAME	Status definition record name
	VER	Status definition record version
RDE	NAME	Record name

Group	Field	Description
	NRDEA	Offset — next RDE
	RECL	Record length (except logical records)
	NLRE	Number of logical record elements
	VER	Record version
	INDX	Relative variable record element index entry
	MINDX	Map record index
	FLG1	Flag byte 1
	FLG2	Flag byte 2
	CRECL	Compressed INIT record size
	INTOF	Offset — RDEINITV within RDE to the compressed initialized record
	NLRA	Offset — next logical record RDE (logical records only)
	FLG3	Flag byte 3
	IMNDX	Input map record index
	OMNDX	Output map record index
	SCHML	Length of schema name when created from an SQL table
	SCHMO	Offset into RDE of schema name when created from an SQL table
	INITV	Initial value (in compressed format)
FDH (format description header)	LEN	Length of format description
	ID	Format identifier
	FDES	Start of format descriptor element
FDE (format description element)	TYPE	Element type
	FLAGS	Flag byte
	PEND	Type dependent section

Group	Field	Description
DCL (declaration module)	NAME	Declaration module name
	VER	Declaration module version
	DATLU	Date module last updated
	DATCR	Date module last created
PME	NAME	Premap process name
	LASTB	Offset of last byte in PME
	RATA	Offset to ready area table
	FCMEA	Offset to first CME ¹
	PVER	Process version
	NCMES	Number of CMEs in response ²
	NEWF	Initialized to X'FF' if new format
	FLAG1	Flag byte
	NMDTE	Number of module table entries
	LNTA	Offset of line number table
	DATLU	Date module last updated
	DATCR	Date module created
	MDTA	Offset of included module table
	OFTBL	Offset to executable code/vector call offset table
RSE	NAME	Response process name
	NXTA	Offset of next RSE in FDB
	LASTB	Offset of last byte in response process
	RATA	Offset to ready area table
	FCMEA	Offset to first CME ¹
	PVER	Process version
	NCMES	Number of CMEs in response ²
	PFKEY	PF key for response
	FUNLN	Length of response field
	OFUNC	Start old-format function code

Group	Field	Description
	FLAG1	Flag byte
	FUNOF	Offset within RSE to function code
	NMDTE	Number of module table entries
	LNTA	Offset of line number table
	DATLU	Date module last updated
	DATCR	Date module created
	MDTA	Offset of included module table
	OFTBL	Offset to executable code/vector call offset table
	FUNC	Response field value
PROCESS OBJECT CODE TABLE	ICMD# GENERATED CODE	Internal command number Executable code and vector calls
CME	CLASS	Command element major class
	FUNC	Command element function
	NXTA	Offset to next CME from first CME within PME or RSE
	NEXT	Offset of next CME with FDB
	INCLUDED MODULE	Name of included module from which CME was generated
	VERS	Included module version
	SEQ#	IDD sequence number
	FLAG1	First flag byte
	FLAG2	Second flag byte
	FLAG3	Third flag byte
	FLAG4	Fourth flag byte
	ICMD#	Internal command number
	BODY	Parameter for use by run-time system
EXEC CODE/ VECTOR CALL OFFSET TABLE	ICMD#	Internal command number

Group	Field	Description
	VECTOR #	Vector code
	CODE/CME LEN	Length of object code, if executable code; length of CME, if a vector call
	CODE OFF	Offset from first item of object code in process
	VECTOR CALL	Identifies an ICMD as a vector call
RAT TABLE		Ready Area Table
INCLUDED MODULE TABLE	PROCESS	Included module name
VER	Included module version	
	DATLU	Date included module last updated
	DATCR	Date included module created
LIT	POOL	Literal pool

¹ If the FDB contains object code, the FCMEA indicates the offset to the first item of object code in the Process Object Code Table.

² If the FDB contains object code, the NCMEs contains the original number of CMEs before their conversion to executable code and vector calls.

Vector call codes The following table lists vector call codes and their associated process commands.

Vector code	Process command
0000	Database command
0001	Database command — logical record
0002	ABORT
0003	INVOKE
0004	TRANSFER
0005	RETURN
0006	DISPLAY
0007	--
0008	LEAVE
0009	LINKT
000A	Assignment command
000B	Conditional command
000C	WHILE REPEAT
000D	Internal branch
000E	Subroutine call
000F	--
0010	ON
0011	ADD
0012	SUBTRACT
0013	MULTIPLY
0014	DIVIDE
0015	MOVE
0016	COMPUTE
0017	MODIFY MAP
0018	(DC) ACCEPT
0019	PUT/GET/DELETE SCRATCH
001A	PUT/GET/DELETE QUEUE
001B	WRITE PRINTER
001C	INITIALIZE RECORDS
001D	KEEP LONGTERM
001E	SNAP
001F	COMMIT

Vector code	Process command
0020	ROLLBACK TASK
0021	EXECUTE NEXT FUNCTION
0022	--
0023	--
0024	Last vector call; end of process
0025	PUT DETAIL
0026	GET DETAIL
0027	WRITE TRANSACTION
0028	READ TRANSACTION
0029	CONTINUE
002A	WRITE TO LOG
002B	CLOSE FILE MAPS
003D	ALLOCATE
003E	CONTROL SESSION
003F	SEND-DATA
	CONFIRM
0040	CONFIRM
0041	CONFIRMED
0042	REQUEST-TO-SEND
0043	SEND-ERROR
0044	RECEIVE-AND-WAIT
0045	PREPARE-TO-RECEIVE
0046	DEALLOCATE
0047	SQL call
0048	TRACE
0049	OPEN
004A	CLOSE
004B	SEND
004C	RECEIVE

Debugging information: The information provided in the dialog reports generated by ADSORPTS can be used for debugging. For example, when the CA-ADS runtime system causes a dialog to abnormally terminate, it sends messages to the system log. The messages provide the following information:

- The reason for the abnormal termination.
- The name of the aborted dialog.
- The name of the process that was executing at the time of the termination.
- The hexadecimal offset within the Fixed Dialog Block (FDB) of the command that was executing at the time of the termination. If the FDB does not contain object code, the offset is to the CME representing the command. If the FDB contains object code, the offset is to the object code that references the CME representing the command.

- The IDD sequence number (SEQ#) of the source line containing the command that caused the abend.
- The internal command number (ICMD#) of the source line containing the command that caused the abend.

Note: The information above is also displayed on the Dialog Abort Information screen, if enabled.

For a discussion of the Dialog Abort Information screen, see Chapter 4, “CA-ADS Runtime System.”

The information in the system messages can be used in conjunction with the FDBLIST report to determine the command that caused the abend. The internal command number and the hexadecimal offset of the problem command can both be used to locate the command as it is represented in the Process Object Code Table, the list of CMEs, and the Executable Code/Vector Call Offset Table. A CME displays the process command that it represents; an item in the Process Object Code Table and the Executable Code/Vector Call Offset Table displays the vector code of the command, as described in the vector call codes table earlier in this appendix.

B.3 Application reports

The application reporting capabilities of ADSORPTS enable the application developer to request any or all of the following reports for one or several applications:

- **Summary reports** list information about task codes, global records, functions, and responses.
- **Records reports** list information about global records.
- **Functions/responses detail reports** list information about functions, responses, and the relationships between functions and responses.
- **Functions/responses summary reports** list the relationships between functions and responses.

Additionally, all of the application reports list basic information about the application, such as security requirements and application-wide defaults.

B.4 Control statements

ADSORPTS is driven by five control statements, as shown below.

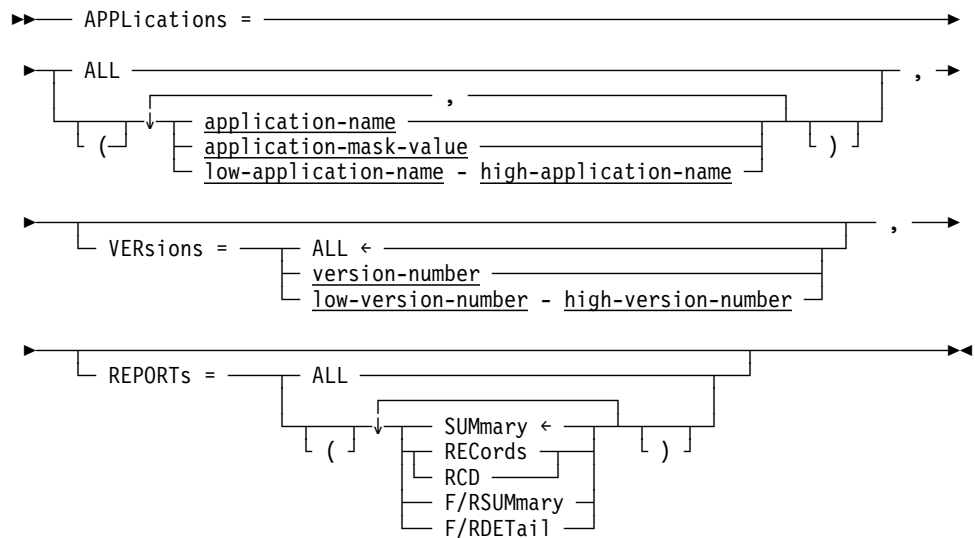
Summary of the ADSORPTS control statements

Control statement	Purpose
APPLICATIONS	Specifies the applications for which reports are being requested and the reports desired for the applications
DIALOGS	Specifies the dialogs for which reports are being requested and the reports desired for the dialogs
LIST	Controls the online or printed format of the ADSORPTS output
SEARCH	Specifies whether ADSORPTS searches in the load (core-image) library or the load area for the dialogs and applications specified by the DIALOGS and APPLICATIONS control statements

B.4.1 APPLICATIONS

Purpose: Generates reports for specified applications.

Syntax:



Parameters

ALL

Specifies all applications in the load area.

application-name

Specifies the 1- to 8-character name of a single application.

If the name includes a hyphen (-) as a character, replace with a mask character.
The mask character is the asterisk (*).

application-mask-value

Specifies any application with a name that matches the mask criteria.

The mask character is the asterisk (*); it matches any character. For example, APPLICATIONS=ORE***** generates the requested reports for all applications beginning with ORE.

low-application-name - high-application-name

Specifies all applications within the application-name range (inclusive).

The hyphen (-) is required and cannot have surrounding blanks.

Application names and masks that have fewer than eight characters are padded on the right with blanks.

VERsions =

Introduces the version numbers of the applications for which reports are requested.

ALL

Specifies all versions of the named applications.

ALL is the default when no other version is specified.

version-number

Specifies a single version number for the named applications.

low-version-number - high-version-number

Specifies all versions of the named applications within the version-number range (inclusive).

The hyphen (-) is required and cannot have surrounding blanks.

REPORTS=

Introduces the reports requested for the named applications.

ALL

Requests the summary, records, and functions/responses detail reports for the named applications.

SUMmary

Requests summary reports for the named applications.

SUMMARY is the default when no other report is specified.

RECords

Requests records reports for the named applications.

F/RSUMmary

Requests functions/responses summary reports for the named applications.

F/RDETail

Requests functions/responses detail reports for the named applications.

Usage

Considerations: A maximum of 100 application report requests can be specified in a single ADSORPTS run. If both dialog and application reports are requested in a single ADSORPTS run, dialogs are reported first, followed by applications.

Examples: Example 1: Requesting summary reports

The following statement requests summary reports for all versions of applications with names in the range A through C (inclusive):

```
APPLICATIONS=(A-C),REPORTS=SUMMARY
```

Example 2: Requesting all reports

The following statement requests all reports for all applications whose names begin with ABC and whose version number is 20:

```
APPLICATIONS=ABC*****,VERSION=20,REPORT=ALL
```

Example 3: Requesting summary and records reports for all versions

The following statement requests summary and records reports for all versions of applications with names that contain the characters S and T in the third and fourth positions and blanks in the last two positions:

```
APPLICATIONS=**ST**,REPORTS=(SUMMARY,RECORDS)
```

Example 4: Requesting functions/responses summary reports

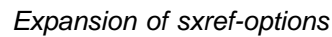
The next statement requests functions/responses summary reports for all versions of the following applications:

- Applications whose names begin with the characters ABC
- Applications whose names contain the characters S and T in the third and fourth positions and blanks in the last two positions
- Applications whose names are in the range A through C (inclusive)

```
APPLICATIONS=(ABC*****,**ST**,A-C),REPORTS=F/RSUMMARY
```

B.4.2 DIALOGS

Syntax



Generates reports for all dialogs in the load area.

Specifies the 1- to 8-character name of a single dialog.

dialog-mask-value

low-dialog-name - high-dialog-name

Specifies all dialogs within the dialog-name range (inclusive).

The hyphen (-) is required and cannot have surrounding blanks.

Dialog names and masks that have fewer than eight characters are padded on the right with blanks.

Introduces the version numbers of the dialogs for which reports are requested.

ALL

Specifies all versions of the named dialogs.

ALL is the default when no other version is specified.

version-number

Specifies a single version number for the named dialogs.

low-version-number - high-version-number

Specifies all versions of the named dialogs within the version-number range (inclusive). The hyphen (-) is required and cannot have surrounding blanks.

REPORTS =

Introduces the reports requested for the named dialogs.

ALL

Requests all reports (that is, the summary, processes, records, and FDBLIST reports) for the named dialogs.

SUMmary

Requests summary reports for the named dialogs.

SUMMARY is the default when no other report is specified.

RECORDs

Requests records reports for the named dialogs.

FDBlist

Requests FDBLIST reports for the named dialogs.

PROcesses

Requests processes reports for the named dialogs.

sxref-options

Specifies sorted cross-reference report options.

with SXREF

Requests a sorted cross-reference for process reports. The usage of all data names and subroutine calls is cross-referenced.

LONG

Specifies that all elements be included in the report.

SHORT

Specifies that only elements that are referenced be included in the report.

SHORT is the default.

Note: When specifying the cross-reference option, the Master Function Table (RHDCEVBF) must reside in either the load area or the load library.

Usage:

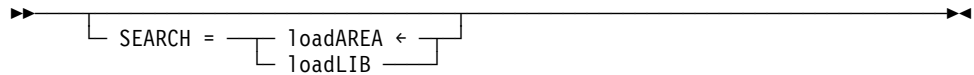
Considerations: A maximum of 100 dialog report requests can be specified in a single ADSORPTS run. If both dialog and application reports are requested in a single ADSORPTS run, dialogs are reported first, followed by applications, regardless of their order in the control statements.

Usage:

Considerations: If more than one LIST statement is submitted for a single run of ADSORPTS, the specification in the last LIST statement applies for the entire run.

B.4.4 SEARCH

Purpose: Specifies where ADSORPTS searches for the object dialogs and applications.

Syntax:**Parameters****loadAREA**

Specifies that ADSORPTS searches for the dialogs and applications in the load area.

LOADAREA is the default when neither LOADAREA OR LOADLIB is specified.

loadLIB

Specifies that ADSORPTS searches for the dialogs and applications in the load (core-image) library.

Usage*Considerations*

- If more than one SEARCH statement is submitted for a single run of ADSORPTS, the specification in the last SEARCH statement applies for the entire run.
- The DIALOGS and APPLICATIONS statements cannot specify a range of dialog or application names (such as *low-dialog-name - high-dialog-name*) or a dialog or application mask (such as *dialog-mask-value*).
- The load (core-image) libraries in which the dialogs and applications are located must be specified in the JCL or commands that run the reports, as follows:
 - **OS/390 JCL**
In the CDMSLIB statement or, if a CDMSLIB statement is not specified, in the STEPLIB statement
 - **VSE/ESA JCL**
In the ASSGN/EXTNT statement for the private core-image library or in the LIBDEF equivalent
 - **VM/ESA commands**
In the GLOBAL LOADLIB command, added to the list of libraries

- **BS2000/OSD JCL** In the CDMSLIB chain.

B.5 SYSIDMS parameter file

See *CA-IDMS Database Administration*, for more information about SYSIDMS parameters.

B.6 JCL and commands to run reports

Sample OS/390 JCL for central version: ADSORPTS (central version) (OS/390)

```
//ADSORPTS EXEC PGM=ADSORPTS,REGION=1024K
//STEPLIB DD DSN=idms.dba.loadlib,DISP=SHR
// DD DSN=idms.loadlib,DISP=SHR
//sysctl DD DSN=idms.sysctl,DISP=SHR
//dcmmsg DD DSN=idms.sysmsg.ddldcmmsg,DISP=SHR
//SYSLST DD SYSOUT=A
//SYSIDMS DD *
DMCL=dmcl-name
DICTNAME=apldict
Put other SYSIDMS parameters, as appropriate, here
/*
//SYSIPT DD *
Put ADSORPTS parameters, as appropriate, here
/*
/*
```

Sample OS/390 JCL for local mode ADSORPTS (local mode) (OS/390)

```
//ADSORPTS EXEC PGM=ADSORPTS,REGION=1024K
//STEPLIB DD DSN=idms.dba.loadlib,DISP=SHR
// DD DSN=idms.loadlib,DISP=SHR
//dictdb DD DSN=idms.apldict.ddldm1,DISP=SHR
//dloddb DD DSN=idms.apldict.ddldclod,DISP=SHR
//dcmmsg DD DSN=idms.sysmsg.ddldcmmsg,DISP=SHR
//sysjrn1 DD DUMMY
//SYSPCH DD syspch-def
//SYSLST DD SYSOUT=A
//SYSIDMS DD *
DMCL=dmcl-name
DICTNAME=apldict
Put other SYSIDMS parameters, as appropriate, here
/*
//SYSIPT DD *
Put ADSORPTS parameters, as appropriate, here
/*
/*
```

<u>idms.dba.loadlib</u>	Data set name of the load library containing the DMCL and database name table load modules
<u>idms.loadlib</u>	Data set name of the load library containing the CA-IDMS executable modules
<u>sysctl</u>	DDname of the SYSCTL file
<u>idms.sysctl</u>	Data set name of the SYSCTL file
<u>dcmmsg</u>	DDname of the system message (DDLDM/ESAG) area
<u>idms.sysmsg.ddldcmmsg</u>	Data set name of the system message (DDLDM/ESAG) area
<u>dmcl-name</u>	Name of the DMCL load module

<u>appldict</u>	Name of the application dictionary
<u>dictdb</u>	DDname of the application dictionary definition (DDLDDL) area
<u>idms.appldict.ddldml</u>	Data set name of the application dictionary definition (DDLDDL) area
<u>dloddb</u>	DDname of the application dictionary definition load (DDLDDL) area
<u>idms.appldict.ddldclod</u>	Data set name of the application dictionary definition load (DDLDCLOD) area
<u>sysjrn1</u>	DDname of the tape journal file

Sample VSE/ESA JCL for central version: ADSORPTS (VSE/ESA)

```
// UPSI      b                if specified in the IDMSOPTI module
// DLBL      userlib
// EXTENT    ,nnnnnn
// LIBDEF    *,SEARCH=(userlib.cdmslib)
// EXEC      ADSORPTS
SYSIDMS parameters
control statements
```

<u>b</u>	appropriate 1- through 8-character UPSI bit switch, as specified in the IDMSOPTI module
<u>nnnnnn</u>	volume serial number of the library
<u>userlib</u>	filename of the CA-IDMS/DB library
<u>userlib.cdmslib</u>	file-id of the CA-IDMS/DB sublibrary
<u>SYSPPH definition</u>	See "ADSA migration syntax considerations" below
<u>SYSIDMS parameters</u>	A list of SYSIDMS parameters for this job

Sample VSE/ESA JCL for local mode: To execute ADSORPTS in local mode, perform the following steps:

1. Remove the UPSI specification.
2. Add the following statements before the EXEC statement:

```
// DLBL      dictdb,'idms.appldict.ddldml',,DA
// EXTENT    sys015,nnnnnn
// ASSGN     sys015,DISK,VOL=nnnnnn,SHR
// DLBL      dloddb,'idms.appldict.ddldclod',,DA
// EXTENT    sys017,nnnnnn
// ASSGN     sys017,DISK,VOL=nnnnnn,SHR
// DLBL      dmsgdb,'idms.system.ddldcmsg',,DA
// EXTENT    sys016,nnnnnn
// ASSGN     sys016,DISK,VOL=nnnnnn,SHR
// TLBL      sys009,'idms.tapejrn1',,nnnnnn,,f
// ASSGN     sys009,TAPE,VOL=nnnnnn
```

<u>idms.appldict.ddldml</u>	= file-id of the data dictionary DDLML area
<u>idms.appldict.ddldclod</u>	= file-id of the data dictionary load area
<u>idms.system.ddldmsg</u>	= file-id of the data dictionary message area
<u>idms.tapejrn1</u>	= file-id of the tape journal file
<u>dictdb</u>	= filename of the data dictionary DDLML area
<u>dloddb</u>	= filename of the data dictionary load area (DDLDCLOD)
<u>dmsgdb</u>	= filename of the data dictionary message area (DDLDM/ESAG)
<u>f</u>	= file number of the tape journal file
<u>nnnnnn</u>	= volume serial number
<u>sys009</u>	= logical unit assignment for the tape journal file
<u>sys015</u>	= logical unit assignment for the data dictionary DDLML area
<u>sys016</u>	= logical unit assignment for the data dictionary message area
<u>sys017</u>	= logical unit assignment for the data dictionary load area

Sample VM/ESA commands for central version: ADSORPTS (VM/ESA)

```
FILEDEF SYSCTL DISK sysctl frame a
FILEDEF SYSLST PRINTER
FILEDEF SYSIDMS DISK sysidms input a
FILEDEF SYSIPT DISK rpts input a
GLOBAL LOADLIB idmslib
OSRUN ADSORPTS
```

<u>sysctl frame a</u>	filename, filetype, and filemode of the SYSCTL file for the CV to run against
<u>sysidms input a</u>	filename, filetype, and filemode of the file containing the SYSIDMS input parameters
<u>rpts input a</u>	file identifier of the file containing ADSORPTS source statements
<u>idmslib</u>	filename of the CA-IDMS/DB LOADLIB library

Sample VM/ESA commands for local mode: To execute ADSORPTS in local mode, add the following commands before the OSRUN command:

```
FILEDEF sysjrn1 TAP1 SL VOLID nnnnnn (RECFM VB LRECL 111 BLKSIZE bbb)
FILEDEF dictdb DISK dictdb addr
FILEDEF dloddb DISK dloddb addr
FILEDEF dmsgdb DISK dmsgdb addr
```

<u>bbb</u>	= block size of the tape journal file
<u>dictdb</u>	= ddname of the data dictionary DDLDDL area
<u>dictdb addr</u>	= disk address of the data dictionary DDLDDL area; for example, 500
<u>dloddb</u>	= ddname of the data dictionary load area (DDLDCLOD)
<u>dloddb addr</u>	= disk address of the data dictionary load area; for example, 500
<u>dmsgdb</u>	= ddname of the data dictionary message area (DDLDM/ESAG)
<u>dmsgdb addr</u>	= disk address of the data dictionary message area
<u>l11</u>	= record length of the tape journal file

Specifying central version or local mode: To specify whether ADSORPTS executes under central version or in local mode, take one of the following actions:

- Specify either `CVMACH=dc/ucf-machine-name` (for central version) or `*LOCAL*` (for local mode) as the first statement to be submitted to ADSORPTS.
Dc/ucf-machine-name is the 1- through 8-character user identifier of the VM/ESA virtual machine in which the CA-IDMS/DC or CA-IDMS/UCF (DC/UCF) system is executing.

- Link edit ADSORPTS with an IDMSOPTI module that specifies either CVMACH=*dc/ucf-machine-name* (for central version) or CENTRAL=NO (for local mode).
- ▶▶ For instructions to create an IDMSOPTI module, refer to *CA-IDMS System Operations*.
- Code PARM='CVMACH=*dc/ucf-machine-name*' or PARM='*LOCAL*' on the OSRUN command used to invoke the compiler. This option is not allowed if the OSRUN command is issued from a VM/ESA EXEC program; however, it is allowed if the OSRUN command is issued from a System Product interpreter (REXX) or EXEC 2 program.

▶▶ For additional information about central version and local mode operations in the VM/ESA environment, refer to *Installation and Maintenance Guide — VM/ESA*.

Sample BS2000/OSD JCL for central version: JCL and commands for running ADSORPTS under central version are shown below for BS2000/OSD JCL:

ADSORPTS (BS2000/OSD)

```
/ADD-FILE-LINK L-NAME=CDMSLIB,F-NAME=idms.dba.loadlib
/ADD-FILE-LINK L-NAME=CDMSLIB1,F-NAME=idms.loadlib
/ADD-FILE-LINK L-NAME=CDMSLODR,F-NAME=idms.loadlib
/ADD-FILE-LINK L-NAME=sysctl,F-NAME=idms.sysctl,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=SYSIDMS,F-NAME=idms.sysidms
/ASSIGN-SYSDTA TO=*SYSCMD
/START-PROG *MOD(ELEM=ADSORPTS,LIB=idms.loadlib,RUN-MODE=*ADV)
```

control statements

<u>idms.loadlib</u>	filename of CA-IDMS/DB load library
<u>idms.dba.loadlib</u>	data set name of the load library containing the DMCL and database name table load modules
<u>idms.sysidms</u>	filename of the file containing the SYSIDMS parameters
<u>idms.sysctl</u>	filename of the SYSCTL file
<u>sysctl</u>	linkname of the SYSCTL file

Sample BS2000/OSD JCL for local mode: To execute ADSORPTS in **local mode**, perform the following steps:

1. Remove the ADD-FILE-LINK statement for sysctl
2. Add the following statements:

```
/ADD-FILE-LINK L-NAME=dictdb,F-NAME=idms.appldict.dictdb,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=dloaddb,F-NAME=idms.appldict.dloaddb,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=dcmsg,F-NAME=idms.sysmsg.ddlccmsg,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=sysjrn1,F-NAME=idms.tapejrn1
```

<u>idms.appldict.ddldml</u>	filename of the data dictionary DDLML area
<u>idms.appldict.ddldclod</u>	filename of the data dictionary load area
<u>idms.system.ddldcmsg</u>	filename of the data dictionary message area
<u>idms.tapejrn1</u>	filename of the tape journal file
<u>dictdb</u>	linkname of the data dictionary DDLML area
<u>dloddb</u>	linkname of the data dictionary load area (DDLDCLOD)
<u>dmsgdb</u>	linkname of the data dictionary message area (DDLDM/ESAG)
<u>sysjrn1</u>	linkname of the tape journal file.

Appendix C. Dialog Statistics

- C.1 Overview C-3
- C.2 Collecting selected statistics C-4
- C.3 Enabling dialog statistics C-8
- C.4 Selecting dialogs C-9
- C.5 Setting a checkpoint interval C-10
- C.6 Collecting and writing statistics C-11
- C.7 Statistics reporting C-12

C.1 Overview

The CA-ADS dialog statistics feature allows collection of runtime statistics about dialog and overhead activity. (Overhead activity is not directly attributable to any dialog. Overhead activity occurs once at the beginning and once at the end of an application.) Statistics are collected for each logical terminal through which the application is executed.

The following aspects of dialog statistics are discussed in this appendix:

- Collecting selected statistics
 - Enabling dialog statistics
 - Selecting dialogs for collection of individual statistics
 - Setting a checkpoint interval, after which accumulated statistics are written to the log file at runtime
 - Collecting and writing statistics at runtime
 - Dialog statistics reporting
- For information about statistics-related DCMT commands described later in this appendix, refer to *CA-IDMS System Tasks and Operator Commands*.

C.2 Collecting selected statistics

Individual sets of dialog statistics can be collected for selected dialogs or for every dialog that executes during an application. If statistics are collected for selected dialogs, one additional set of statistics is collected for all the nonselected dialogs.

Transaction Statistics Block fields: The following table lists the sets of CA-IDMS/DB and CA-IDMS/DC transaction statistics that can be collected for each dialog and for overhead activity.

►► For information about transaction statistics, refer to *CA-IDMS System Operations*.

Type of information	Fields
IDENTIFICATION INFORMATION	Transaction Statistics Block identifier DC user identifier DC logical terminal identifier Dialog identifier Date that BIND command was issued Time that BIND command was issued
CA-IDMS/DC STATISTICS	Number of programs called Number of programs loaded Number of terminal reads Number of terminal writes Number of terminal errors Number of storage acquisitions Number of scratch gets Number of scratch puts Number of scratch deletes Number of queue gets Number of queue puts Number of queue deletes Number of get time requests Number of set time requests Number of database calls Max words used in stack User mode time System mode time Wait time Task storage high-water mark Total number of free storage requests

Type of information	Fields
IDMS-DB STATISTICS	Number of pages read Number of pages written Number of pages requested Number of CALC records stored with no overflow Number of CALC records stored with overflow Number of VIA records stored with no overflow Number of VIA records stored with overflow Number of records requested Number of records current of run unit Number of fragments stored Number of records relocated Total number of locks Number of select locks Number of update locks

CA-ADS Statistics Block fields: The following table lists the sets of CA-ADS statistics that can be collected for each dialog.

Type of information	Fields
IDENTIFICATION INFORMATION	CA-ADS Statistics Block identifier DC user identifier DC logical terminal identifier Dialog identifier Date that Transaction Statistics Block BIND command was issued Time that Transaction Statistics Block BIND command was issued Dialog version number

Type of information	Fields
STATISTICS FOR EXPLICITLY CODED CONTROL COMMANDS	Number of DISPLAY commands Number of DISPLAY CONTINUE commands Number of INVOKE commands Number of LINK TO DIALOG commands Number of LINK TO PROGRAM commands Number of RETURN commands Number of RETURN CONTINUE commands Number of TRANSFER commands Number of LEAVE ADS commands Number of LEAVE APPLICATION commands Number of ABORT commands
STATISTICS FOR IMPLICITLY GENERATED CONTROL COMMANDS	Number of DISPLAY commands Number of INVOKE commands Number of LINK TO DIALOG commands Number of LINK TO PROGRAM commands Number of RETURN commands Number of RETURN CONTINUE commands Number of TRANSFER commands Number of LEAVE ADS commands Number of LEAVE APPLICATION commands Number of ABORT commands

Type of information	Fields
DIALOG EXECUTION STATISTICS	Number of premap process executions Number of response process executions Number of statistics accumulation calls Number of explicit scratch gets Number of explicit scratch puts Number of explicit scratch deletes Number of WRITE PRINTER commands Number of PUT NEW DETAIL commands Number of PUT CURRENT DETAIL commands Number of GET DETAIL commands Size of Fixed Dialog Block (FDB) Size of Variable Dialog Block (VDB) Highest link level at which a dialog was executed Lowest link level at which a dialog was executed
STATISTICS FOR RECORD BUFFER BLOCK (RBB) USAGE	Number of times RBBs put to scratch Most RBB storage used (all dialogs) RBB free space when most storage used Least RBB storage used (all dialogs) RBB free space when least storage used Most RBB space acquired for a dialog Least RBB space acquired for a dialog Highest number of RBBs used Lowest number of RBBs used

C.3 Enabling dialog statistics

Dialog statistics can be collected only if task and transaction statistics collection is enabled. Task statistics are enabled at system generation time. Transaction statistics can be enabled at either system generation or runtime in the following manner:

- To enable transaction statistics at **system generation time**, use the STATISTICS parameter in the SYSTEM statement, specifying TASK, WRITE, and TRANSACTION.
- To enable transaction statistics at **runtime**, use the DCMT VARY STATISTICS TRANSACTION command.

The DCMT VARY ADSO STATISTICS command is used to enable or disable dialog statistics and to specify whether individual statistics are collected for selected dialogs or for all dialogs, as follows:

- **DCMT VARY ADSO STATISTICS ON ALL DIALOGS** enables dialog statistics and specifies that sets of statistics are to be collected for overhead activity and for each dialog that is executed during a CA-ADS application.
- **DCMT VARY ADSO STATISTICS ON SELECTED DIALOGS** enables dialog statistics and specifies that sets of statistics are to be collected for overhead activity and for each selected dialog that is executed during an CA-ADS application. One additional set of statistics is collected to accumulate statistics for all nonselected dialogs.
- **DCMT VARY ADSO STATISTICS OFF** disables the dialog statistics feature.

If no DCMT VARY ADSO STATISTICS command is issued prior to the execution of an application, the runtime system uses the default specification established at system generation.

C.4 Selecting dialogs

The DCMT VARY PROGRAM command is used to select or deselect dialogs for individual statistics collection, as follows:

- **DCMT VARY PROGRAM *dialog-name* ADSO STATISTICS ON** selects the named dialog for individual statistics collection. At runtime, if dialog statistics are enabled, individual statistics are collected for the dialog when it executes.
- **DCMT VARY PROGRAM *dialog-name* ADSO STATISTICS OFF** deselects a dialog from individual statistics collection. At runtime, if dialog statistics are enabled in the SELECTED DIALOGS mode, individual statistics for the dialog are not collected; however, one set of statistics is collected for all nonselected dialogs. If dialog statistics are enabled in the ALL DIALOGS mode, individual statistics are collected for the dialog when it executes, even if it is not selected.

If no DCMT VARY PROGRAM command with the STATISTICS parameter is issued for a dialog prior to the execution of an application, the runtime system uses the specification established at system generation.

The DCMT VARY ADSO STATISTICS and the DCMT VARY PROGRAM commands can be issued in any order.

C.5 Setting a checkpoint interval

The DCMT VARY ADSO STATISTICS command is used to set a checkpoint interval, which determines when the collected statistics are written to the system log, as follows:

- **DCMT VARY ADSO STATISTICS CHECKPOINT INTERVAL**
checkpoint-interval-number specifies that statistics for all dialogs are written to the log once at every *checkpoint-interval-number* statistics accumulations. Additionally, statistics are written to the log when the application terminates. Note that CHECKPOINT INTERVAL 0 is equivalent to CHECKPOINT INTERVAL OFF.
- **DCMT VARY ADSO STATISTICS CHECKPOINT INTERVAL OFF**
specifies that statistics are written to the log only when the application terminates.

If no DCMT VARY ADSO STATISTICS command with the CHECKPOINT INTERVAL parameter is issued prior to the execution of an application, the runtime system uses the specification established at system generation.

C.6 Collecting and writing statistics

At runtime, if dialog statistics are enabled, statistics for overhead activity are collected and written to the CA-IDMS/DC or CA-IDMS/UCF (DC/UCF) system log whenever overhead activity is performed, once at the beginning of the application and once at the end. The transaction statistics block identifier for overhead activity is either the application name or, for applications not defined using the application generator, \$ADS@@OH.

Dialog statistics are collected each time a dialog issues a control command. These statistics are not written immediately to the system log, but are accumulated in transaction and CA-ADS statistics blocks (TSBs and ASBs).

The runtime system allocates TSBs and ASBs as follows:

- If dialog statistics are enabled in the ALL DIALOGS mode, one TSB and one ASB are allocated for each dialog the first time the dialog becomes operative in the application thread. The statistics block identifier for the TSB and ASB is the dialog name.
- If dialog statistics are enabled in the SELECTED DIALOGS mode, one TSB and one ASB are allocated for each selected dialog the first time the dialog becomes operative in the application thread. Additionally, one TSB and one ASB are allocated to accumulate statistics for all nonselected dialogs; the statistics block identifier for the additional TSB and ASB is \$ADS@@AO.

Dialog statistics are written to the system log each time the number of statistics accumulations equals the predefined checkpoint interval. Additionally, statistics are written to the log when the application terminates.

When dialog statistics are written to the system log, only TSBs and ASBs that contain accumulated statistics are written to the system log. The TSBs and ASBs are then initialized, and the statistics accumulations count is reset to zero.

TSBs and ASBs are freed only when the application terminates. Note, however, that during a pseudo-converse they may be written to scratch along with record buffer blocks, as directed at system generation with the FAST MODE THRESHOLD and RESOURCES parameters of the ADSO statement.

C.7 Statistics reporting

DC/UCF statistics reports (SREPORTs) allow the application developer to produce reports on dialog statistics.

Statistics collected in the CA-ADS statistics block can be reported on by using any of the following SREPORTs, identified by report number:

- **018**— CA-ADS statistics by user id
- **019**— CA-ADS statistics by dialog and version number
- **020**— CA-ADS statistics by logical terminal id

Statistics collected in the transaction statistics block can be reported on by using any of the following SREPORTs:

- **011**— CA-IDMS-DC transaction statistics by logical terminal id
- **021**— IDMS-DC transaction statistics by dialog and

SREPORTS are similar in format.

Sample SREPORT for dialog statistics: The following shows sample output from SREPORT number 019:

REPORT NO. 019		ADS STATISTICS BY DIALOG AND VERSION NUMBER - R15.0		09/19/99	PAGE 8
DIALOG NAME : AD5OAFNC		VERSION NUMBER: 1			
DATE :	91043 TIME :	09:49 USER ID :	SMT		
DATE BIND :	91043 TIME BIND :	09:46 LTERM ID :	LT12011		
DISPLAY COMMAND:	21 DISPLAY CONTINU:	21 INVOKES :	3 LINK TO DIALOGS:	18	
LNKS TO PROGRAM:	18 RETURNS :	0 RETURN CONTINUE:	0 TRANSFERS :	18	
LEAVE ADS :	0 LEAVE APPLICATN:	0 ABORTS :	0 IMPL DISPLAYS :	0	
IMPL INVOKE :	0 IMPL LINK DLGS :	0 IMPL LINK PGMS :	0 IMPL RETURNS :	0	
IMPL RET CONT :	0 IMPL TRANSFERS :	0 IMPL LEAVE ADS :	0 IMPL LEAVE PGMS:	0	
IMPL ABORTS :	0 PREMAP PROCESS :	42 RESPONSE PROCES:	21 STAT ACCUM CALL:	99	
EXPL GET SCRS :	0 EXPL PUT SCRS :	0 EXPL DEL SCRS :	0 WRTE PRINT REQS:	0	
PUT NEW DETAILS:	0 PUT CUR DETAILS:	0 GET DETAILS :	0 SIZE OF FDB :	23,080	
SIZE OF VDB :	836 HIGHEST LNK LEV:	1 LOWEST LNK LEVL:	1 RBB PUT TO SCR :	0	
RBB STG HI MARK:	3,176 RBB FREE HI :	908 RBB STG LOW MK :	3,176 RBB FREE LOW :	908	
MOST RBB ACQ :	304 LEAST RBB ACQ :	304 HICOUNT RBB USE:	1 LOCOUNT RBB USE:	1	
**** DIALOG TOTAL ****					
DISPLAY COMMAND:	21 DISPLAY CONTINU:	21 INVOKES :	3 LINK TO DIALOGS:	18	
LNKS TO PROGRAM:	18 RETURNS :	0 RETURN CONTINUE:	0 TRANSFERS :	18	
LEAVE ADS :	0 LEAVE APPLICATN:	0 ABORTS :	0 IMPL DISPLAYS :	0	
IMPL INVOKE :	0 IMPL LINK DLGS :	0 IMPL LINK PGMS :	0 IMPL RETURNS :	0	
IMPL RET CONT :	0 IMPL TRANSFERS :	0 IMPL LEAVE ADS :	0 IMPL LEAVE PGMS:	0	
IMPL ABORTS :	0 PREMAP PROCESS :	42 RESPONSE PROC :	21 STAT ACCUM CALL:	99	
EXPL GET SCRS :	0 EXPL PUT SCRS :	0 EXPL DEL SCRS :	0 WRTE PRINT REQS:	0	
PUT NEW DETAILS:	0 PUT CUR DETAILS:	0 GET DETAILS :	0 RECORD COUNT :	1	

Appendix D. Application and Dialog Utilities

D.1 Overview	D-3
D.2 ADSOBCOM	D-4
D.2.1 Standard control statements	D-4
D.2.2 Special control statements	D-5
D.2.3 SIGNON	D-5
D.2.4 COMPILE	D-6
D.2.5 DECOMPILE	D-8
D.2.6 Dialog-expression	D-10
D.2.7 JCL and commands	D-30
D.2.7.1 OS/390 JCL	D-30
D.2.7.2 VSE/ESA JCL	D-31
D.2.7.3 VM/ESA commands	D-33
D.2.7.4 BS2000/OSD JCL	D-35
D.3 ADSOBSYS	D-37
D.3.1 Control statements	D-37
D.3.2 SYSTEM statement	D-38
D.3.3 JCL and commands	D-39
D.3.3.1 OS/390 JCL	D-39
D.3.3.2 VSE/ESA JCL	D-42
D.3.3.3 VM/ESA commands	D-44
D.3.3.4 BS2000/OSD JCL	D-46
D.4 ADSOBTAT	D-48
D.4.1 Control statements	D-49
D.4.2 JCL and commands	D-51
D.4.2.1 OS/390 JCL	D-51
D.4.2.2 VSE/ESA JCL	D-52
D.4.2.3 VM/ESA commands	D-54
D.4.2.4 BS2000/OSD JCL	D-55
D.5 ADSOTATU	D-57
D.5.1 TAT update utility screen	D-58

D.1 Overview

CA-ADS provides utilities that allow the application developer to maintain applications and dialogs. The utilities are summarized in the table below and discussed in further detail throughout this appendix.

Summary of CA-ADS utilities

Utility	Purpose
ADSOBCOM	Creates, modifies, deletes, and recompiles dialogs in batch mode
ADSOBSYS	Sets up system generation parameters required by ADSOBCOM
ADSOBTAT	Modifies the task application table (TAT) in batch mode when an application is migrated from one dictionary to another
ADSOTATU	Modifies the task application table (TAT) online when an application is migrated from one dictionary to another

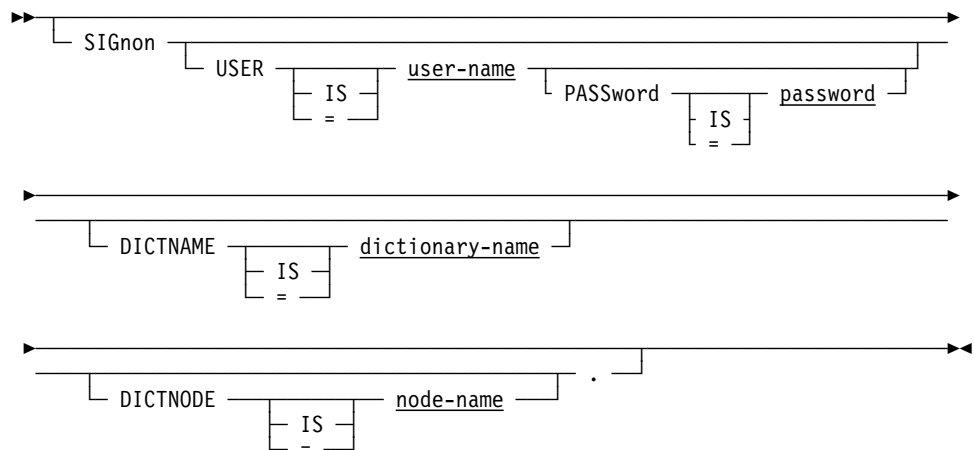
D.2.2 Special control statements

ADSOBCOM is driven by the control statements SIGNON, COMPILE, and DECOMPILE.

D.2.3 SIGNON

Purpose Specifies the name and any necessary password of the DC/UCF user, as well as the dictionary in which the dialogs to be recompiled are stored.

Syntax:



Parameters

USER is user-name

Specifies the signon user.

The equals sign (=) can be used in place of IS.

Note: USER must be the first parameter specified on the SIGNON statement.

PASSword is password

Specifies, when necessary, the user's DC/UCF password.

The user name and password must be supplied in order to use ADSOBCOM when dialog compiler level security is in effect. Additionally, security at the dialog level may also require that the user name and password be supplied.

►► For a discussion of dialog compiler security, see Appendix G, “Security Features.”

DICTNAME is dictionary-name

Specifies the 1- to 8-character name of the data dictionary from which the dialog load module, process source code, record, map, and subschema definitions are retrieved. This is the same dictionary into which the compiled dialog load module is placed.

If no dictionary name is specified, ADSOBCOM uses the name of the primary dictionary.

DICTNODE is node-name

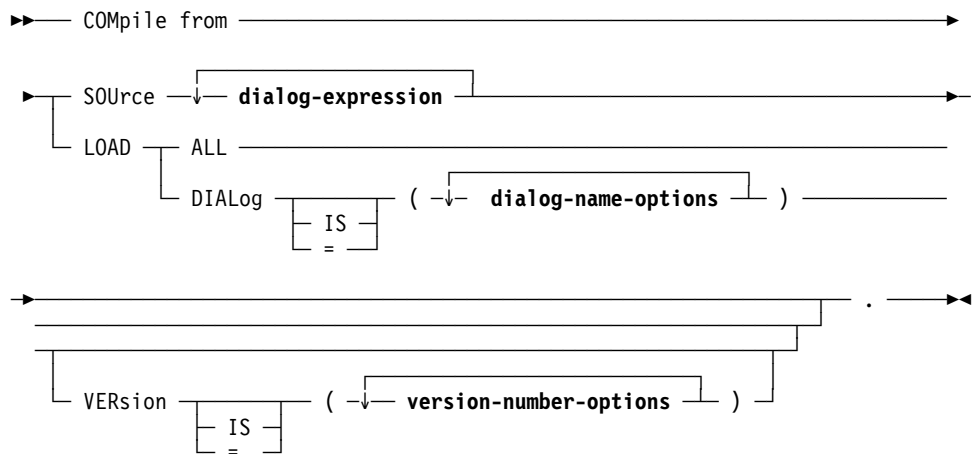
(for DDS only) Specifies the 1- to 8-character name of the DDS node that controls the data dictionary specified by DBNAME.

D.2.4 COMPILE

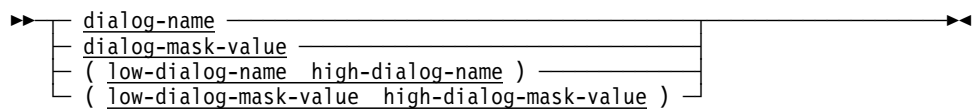
Purpose: Either specifies the dialogs to be recompiled based on information in the load module, or specifies the dialogs to be added, modified, or deleted, based on information in the dialog statements that accompany the COMPILE statement.

There is no limit to the number of COMPILE statements that can be submitted to each run of ADSOBCOM.

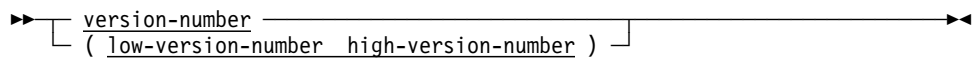
Syntax:



Expansion of dialog-name-options



Expansion of version-number-options



Parameters

SOURCE dialog-expression

Specifies that dialogs to be added, modified, or deleted based on information in the dialog expression.

Repeated dialog expressions can be used to process several dialogs. Each expression must end with a period.

See the explanation of dialog-expression on the following pages.

LOAD

Specifies that the dialogs are to be recompiled based on the information in the dialog load modules.

ALL

Specifies that all dialogs in the dictionary load area are to be recompiled.

DIALog is dialog-name-options

Specifies the dialogs in the dictionary load area to be recompiled. See expansion of dialog-name-options below.

VERsion is version-number-options

Specifies the version numbers of the dialogs to be recompiled. See expansion of version-number-options below.

dialog-name

Specifies the 1 to 8-character name of a single dialog.

dialog-mask-value

Specifies any dialog with a name that matches the mask criteria.

The mask character is the asterisk (*); it matches any character. For example, DIALOG IS (DCB*****) causes all dialogs beginning with DCB to be recompiled.

If the mask contains fewer than eight characters, the remaining character positions are treated as blanks.

(low-dialog-name high-dialog-name)

Specifies all dialogs within the dialog-name range (inclusive).

Note: Parentheses are needed when using a range of values.

(low-dialog-mask-value high-dialog-mask-value)

Specifies all dialogs within the dialog-mask range (inclusive).

Note: Parentheses are needed when using a range of values.

version-number

Specifies a single version number for the selected dialogs.

(low-version-number high-version-number)

Specifies all versions of the selected dialogs within the version-number range (inclusive).

Note: Parentheses are needed when using a range of values.

The default version number is 1.

Usage:

Considerations: ADSOBCOM does not update a dialog's program definition element (PDE) to indicate that a new copy of the dialog exists in the load area. If a dialog is recompiled by ADSOBCOM and then executed during a single DC/UCF run, the application developer should update the PDE by issuing the following command:

DCMT VARY PROGRAM dialog-name NEW COPY

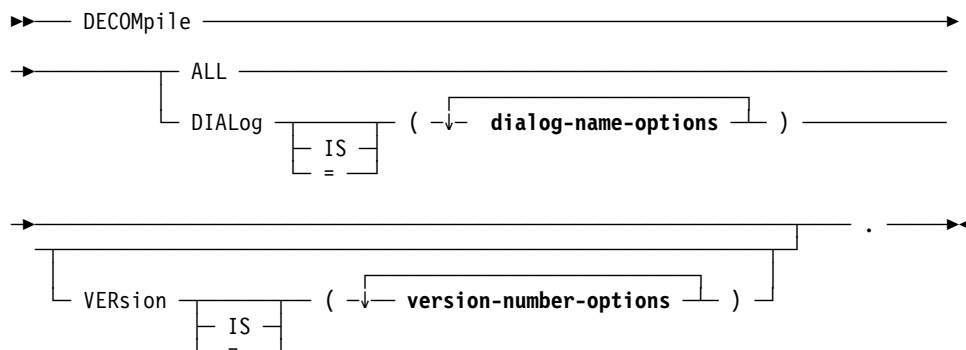
For more information on the DCMT VARY PROGRAM command, refer to *CA-IDMS System Tasks and Operator Commands*.

D.2.5 DECOMPILE

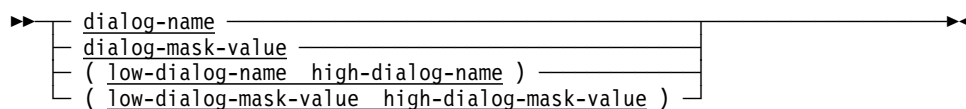
Purpose: Specifies the dialogs to be decompiled based on information in the load module.

There is no limit to the number of DECOMPILE statements that can be submitted to each run of ADSOBCOM.

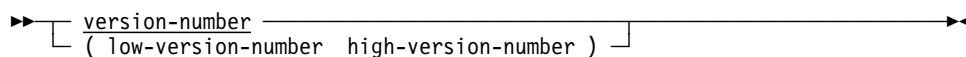
Syntax:



Expansion of dialog-name-options



Expansion of version-number-options



Parameters

SOURCE dialog-expression

Specifies that dialogs to be added, modified, or deleted based on information in the dialog expression.

Repeated dialog expressions can be used to process several dialogs. Each expression must end with a period.

See the explanation of dialog-expression on the following pages.

LOAD

Specifies that the dialogs are to be recompiled based on the information in the dialog load modules.

ALL

Specifies that all dialogs in the dictionary load area are to be recompiled.

DIALog is dialog-name-options

Specifies the dialogs in the dictionary load area to be recompiled. See expansion of dialog-name-options below.

VERsion is version-number-options

Specifies the version numbers of the dialogs to be recompiled. See expansion of version-number-options below.

dialog-name

Specifies the 1- to 8-character name of a single dialog.

dialog-mask-value

Specifies any dialog with a name that matches the mask criteria.

The mask character is the asterisk (*); it matches any character. For example, DIALOG IS (DCB*****) causes all dialogs beginning with DCB to be recompiled.

If the mask contains fewer than eight characters, the remaining character positions are treated as blanks.

(low-dialog-name high-dialog-name)

Specifies all dialogs within the dialog-name range (inclusive).

Note: Parentheses are needed when using a range of values.

(low-dialog-mask-value high-dialog-mask-value)

Specifies all dialogs within the dialog-mask range (inclusive).

Note: Parentheses are needed when using a range of values.

version-number

Specifies a single version number for the selected dialogs.

(low-version-number high-version-number)

Specifies all versions of the selected dialogs within the version-number range (inclusive).

Note: Parentheses are needed when using a range of values.

The default version number is 1.

Usage:

Considerations: ADSOBCOM does not update a dialog's program definition element (PDE) to indicate that a new copy of the dialog exists in the load area. If a dialog is recompiled by ADSOBCOM and then executed during a single DC/UCF run, the application developer should update the PDE by issuing the following command:

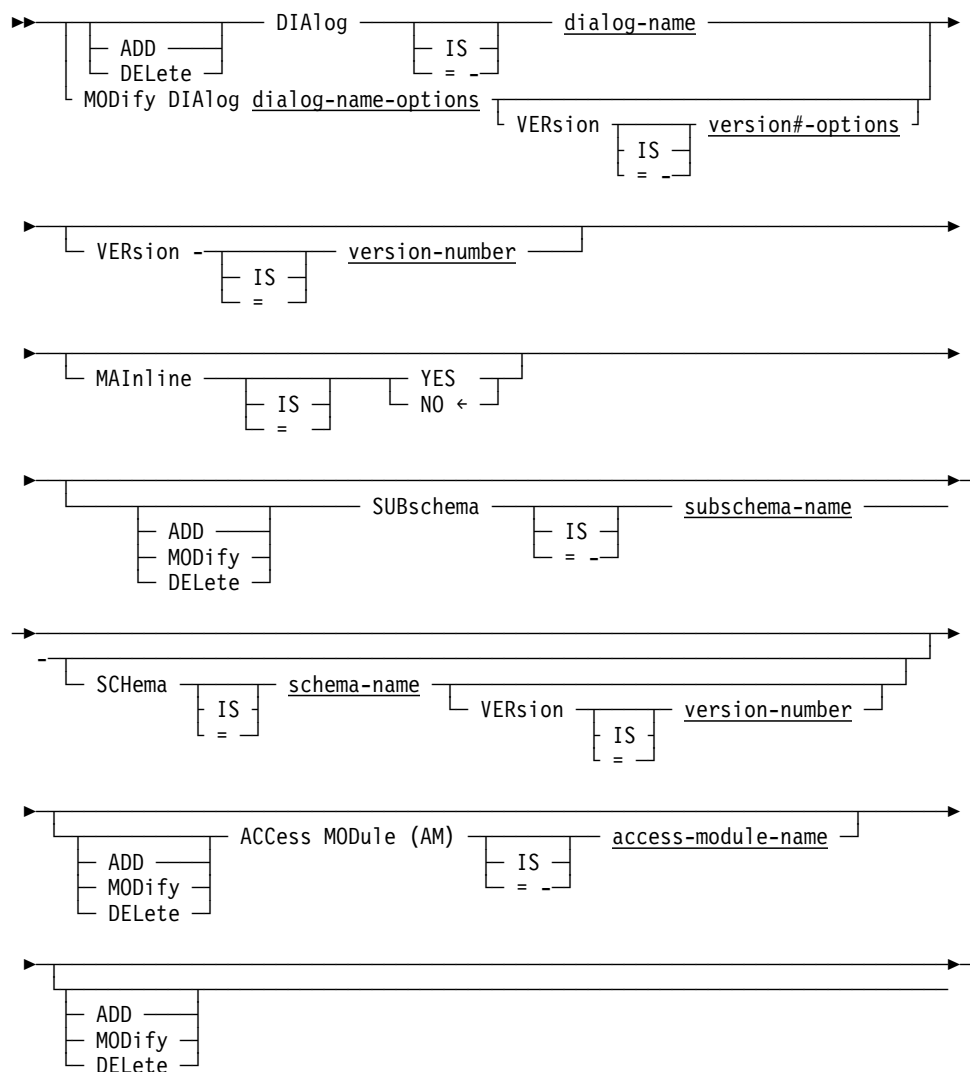
DCMT VARY PROGRAM dialog-name NEW COPY

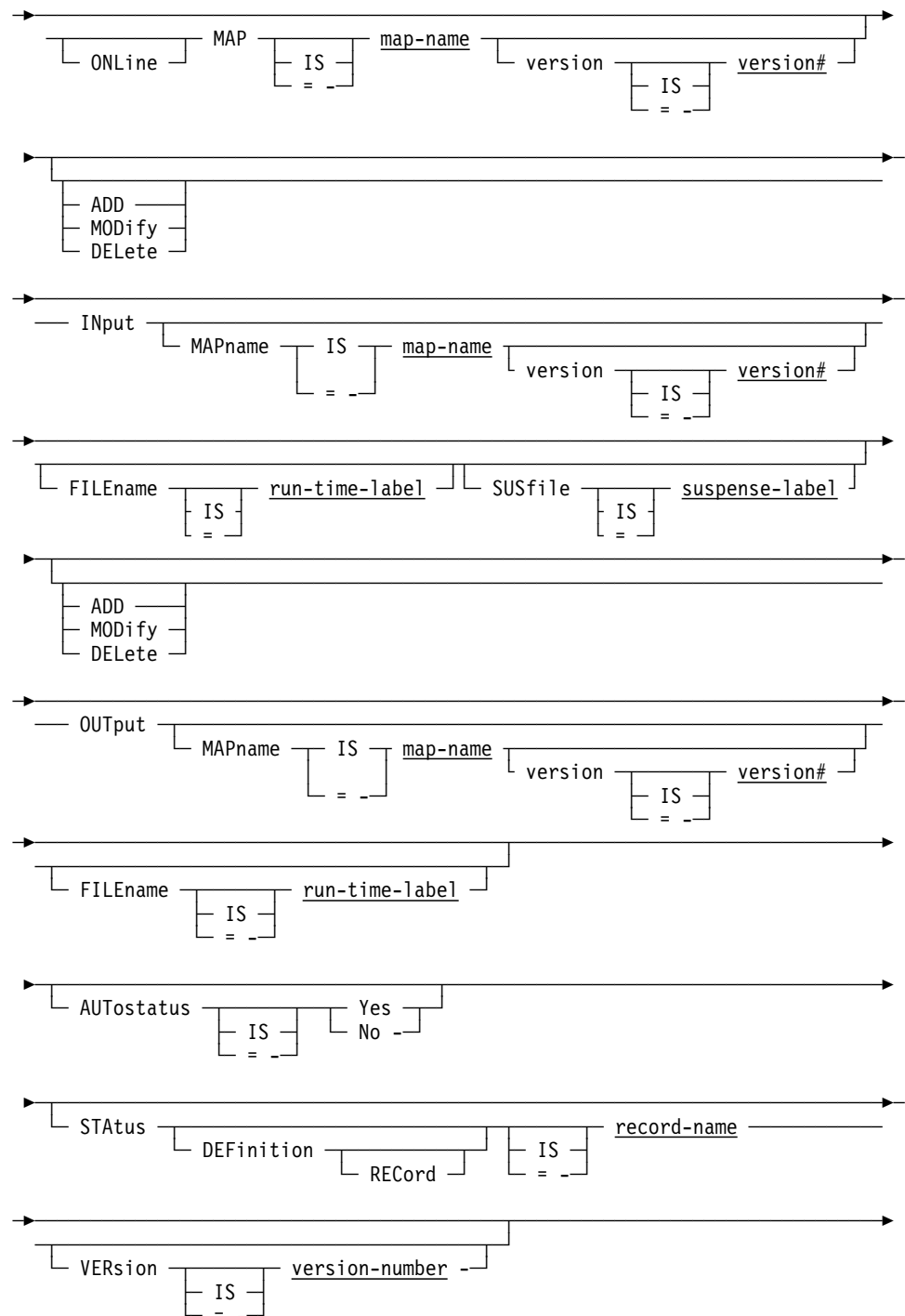
For more information on the DCMT VARY PROGRAM command, refer to *CA-IDMS System Tasks and Operator Commands*.

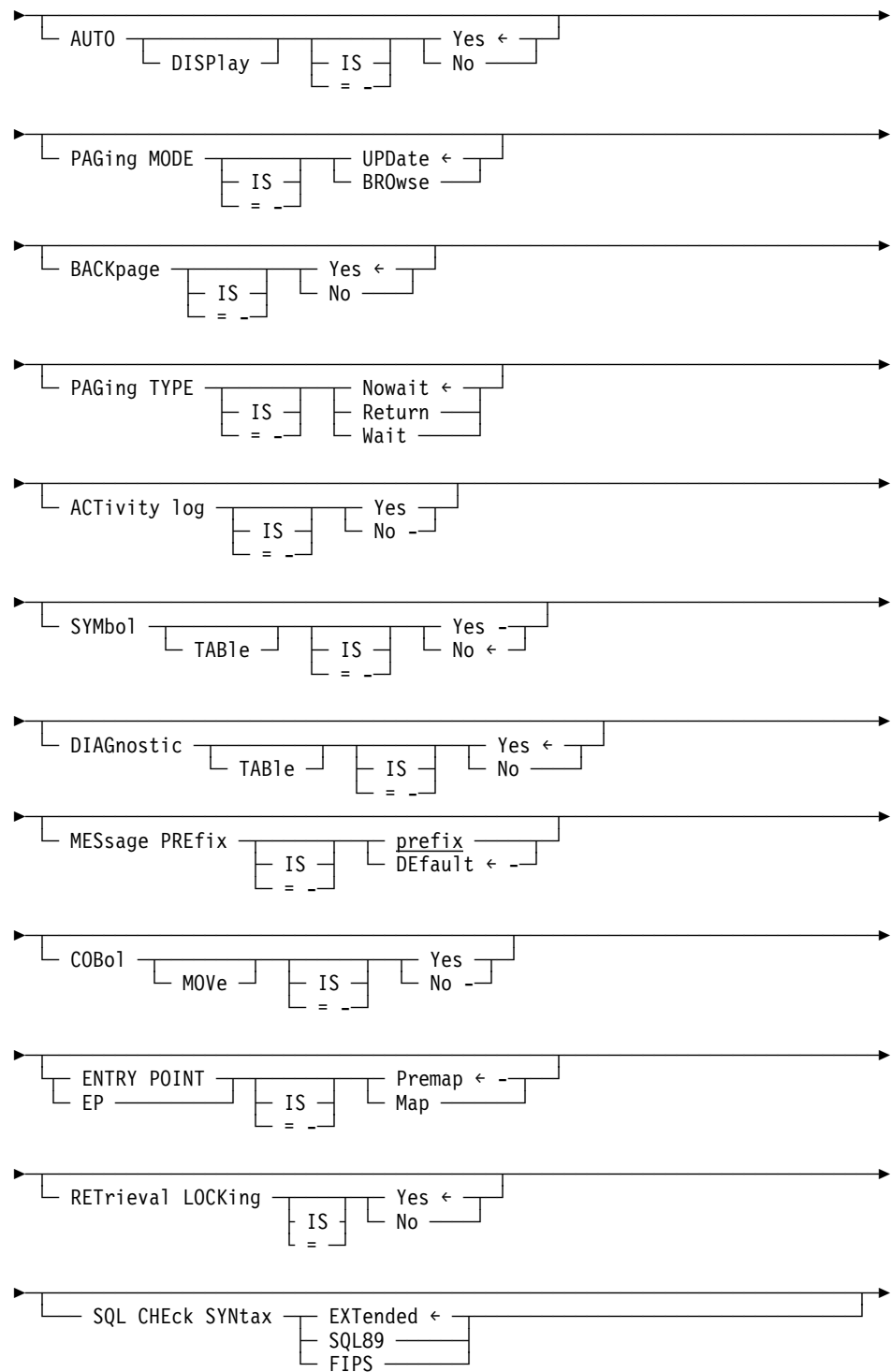
D.2.6 Dialog-expression

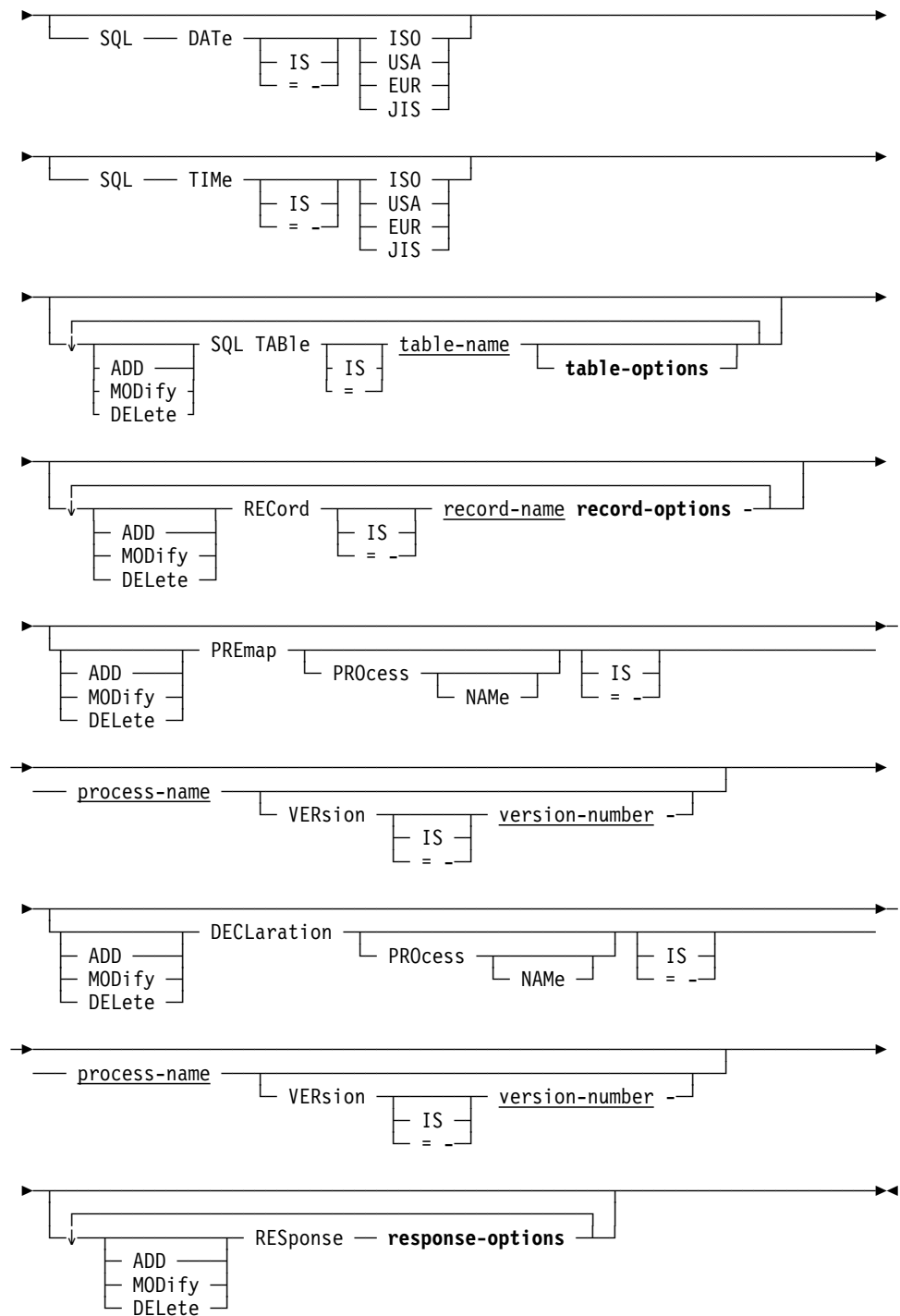
Purpose: *Dialog-expression* is used to specify those dialogs which are to be added, modified, and deleted.

Syntax:

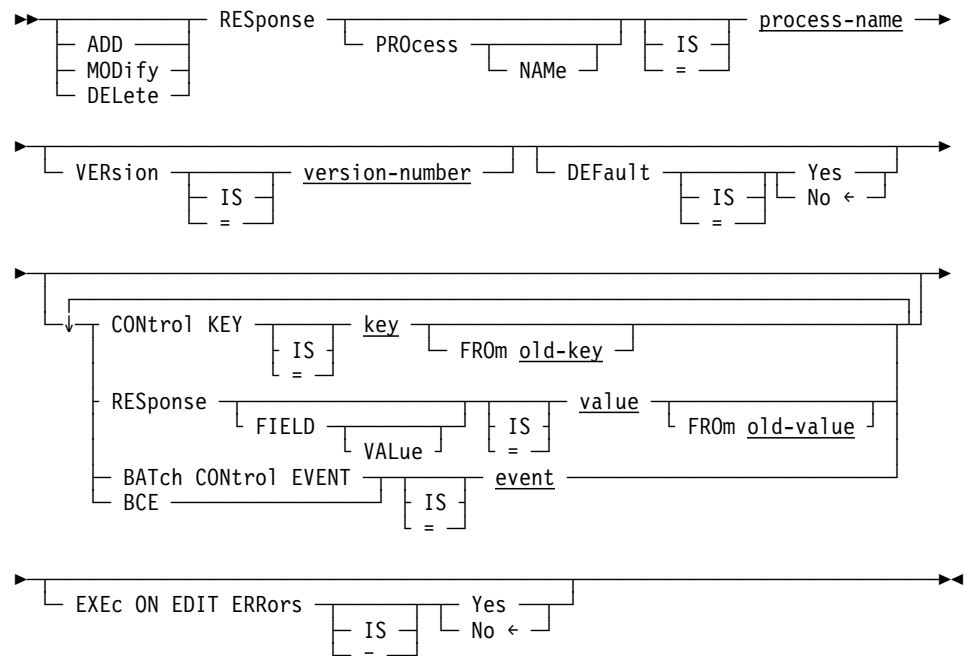




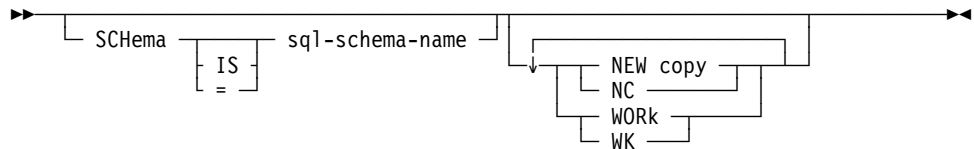




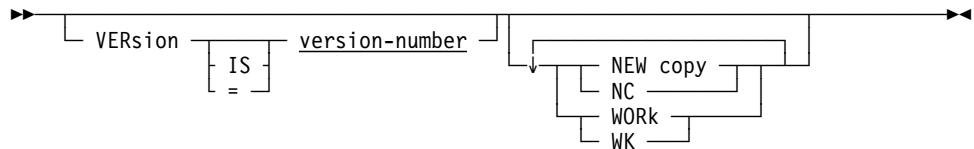
Expansion of response-options



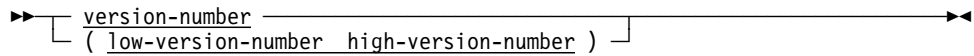
Expansion of table-options



Expansion of record-options



Expansion of version#-options



Parameters

ADD

Specifies that a dialog is to be added to the data dictionary.

ADD is the default if the named dialog does not exist in the data dictionary.

MODify

Specifies that an existing dialog is to be modified.

MODIFY is the default if the named dialog exists in the data dictionary.

DELeTe

Specifies that an existing dialog is to be deleted.

When the action is DELETE, only the dialog name and version number can be specified in the dialog expression.

DIAlog is dialog-name

Specifies the 1- to 8-character name of the dialog being added, modified, or deleted.

The dialog name must begin with an alphabetic or national (@, #, and \$) character and cannot contain embedded blanks.

The equals sign (=) can be used in place of IS.

VERsion is version-number

Specifies the version number (in the range 1 through 9999) of the dialog being added, modified, or deleted.

The default version number is 1.

The equals sign (=) can be used in place of IS.

MAInline is Yes/No

Specifies whether the dialog is a mainline dialog.

At runtime, the dialog that executes first in a series of dialogs that make up an application must be a mainline dialog. If a dialog function is initiated by an application task code, the dialog associated with the function must be a mainline dialog.

►► For more information on mainline dialogs, see Chapter 15, “Control Commands.”

No is the default when neither Yes or No is specified.

ADD

Specifies that the subschema specification is to be added.

ADD is the default if no subschema is associated with the dialog.

MODify

Specifies that the existing subschema specification is to be replaced by a new subschema specification.

MODIFY is the default if a subschema is associated with the dialog.

DELeTe

Specifies that the subschema specification is to be deleted.

If the action is DELETE, the SCHEMA clause cannot be specified.

SUBSchema is subschema-name

Specifies the 1- to 8-character name of the subschema associated with the dialog.

The equals sign (=) can be used in place of IS.

The specified subschema must be defined in the data dictionary. If no subschema is specified for a dialog, the dialog cannot perform database access.

SCHEMA is schema-name

Specifies the 1- to 8-character name of the schema.

A schema name must be specified if the named subschema is associated with more than one schema or version of a schema. If the named subschema is associated with only one schema and version, SCHEMA defaults to the name of that schema.

The equals sign (=) can be used in place of IS.

VERSION is version-number

Specifies the version number (in the range 1 through 9999) of the named schema.

The equals sign (=) can be used in place of IS.

If no version number is specified, VERSION defaults to the version of the named schema that was defined most recently.

ADD

Specifies that the access module specification is to be added.

MODIFY

Specifies that the existing access module specification is to be replaced by a new access module specification.

DELETE

Specifies that the access module specification is to be cleared to spaces.

ACCESS MODULE

Sets the access module name which is used at runtime to satisfy the IDMS/DB request of the dialog.

(AM) is access-module-name

Specifies the 1- to 8-character name of the access module associated with the current dialog.

The dialog can override this specification at runtime by issuing a SET ACCESS MODULE statement.

When the access module name is not specified, the name defaults to the dialog name.

►► For more information about specifying access modules, see 3.3.4, "Database Specifications screen."

ADD

Specifies that a map specification is to be added to the dialog.

ADD is the default if no map of the type specified (online, input, or output) is associated with the dialog.

MODIFY

Specifies that the existing map specification is to be replaced by a new map specification.

MODIFY is the default if a map of the type specified (online, input, or output) is already associated with the dialog.

DELeTe

Specifies that the map definition is to be dissociated from the dialog.

If DELETE is specified, the version number of the MAPNAME clause cannot be specified.

ONLine/INput/OUTput

Specifies the type of map.

ONLINE is the default when no other map type is specified.

A dialog associated with an online map cannot be associated with an input or output file map. A dialog can be associated with both an input and an output file map by coding multiple ADD ... MAPNAME clauses. A dialog not associated with a map is called a mapless dialog and can be executed in both batch and online environments.

MAPname is map-name

Specifies the 1- to 8-character name of the map associated with the dialog.

The specified map must be defined in the data dictionary; however, the map load module does not have to exist. If the dialog has no map specification, only a premap process (not a response process) can be associated with the dialog.

VERsion is version-number

Specifies the version number (in the range 1 through 9999) of the named map.

The equals sign (=) can be used in place of IS.

If no version number is specified and the map is being added to a dialog, or the dialog is being associated with a different map, the version defaults to 1.

Otherwise, the version number defaults to the version of the map currently associated with the dialog.

FILEname is runtime-label

(Batch only) Specifies the OS/390 ddname (VSE/ESA filename, BS2000/OSD linkname, VM/ESA ddname) of the input or output file added or modified.

The equals sign (=) can be used in place of IS.

The runtime label must be specified either during dialog definition or at runtime.

The runtime control statement overrides the default specified during dialog definition.

SUSfile is suspense-label

(Batch only) Specifies the OS/390 ddname (VSE/ESA filename, BS2000/OSD linkname, VM/ESA ddname) of the suspense file for input file maps only.

The equals sign (=) can be used in place of IS.

If a suspense file is maintained for the dialog at runtime, the label must be specified either during dialog definition or at runtime. The runtime control statement overrides the default specified during dialog definition.

AUTostatus is Yes/No

Specifies whether the autostatus facility is used when the current dialog executes.

The default setting corresponds to the autostatus specification defined at DC/UCF system generation. If autostatus is defined as optional, the application developer can override the initial setting. If autostatus is defined as mandatory, the initial setting cannot be changed.

►► For a discussion of the autostatus facility, see Chapter 10, “Error Handling.”

STATus DEFinition RECord is record-name

Specifies the 1 to 32-character name of the status definition record.

The specified record must be defined in the data dictionary. If no record name is specified, STATUS DEFINITION RECORD defaults to the name of the status definition record defined at DC/UCF system generation.

►► For more information on status definition records, see Chapter 10, “Error Handling.”

VERsion is version-number

Specifies the version number (in the range 1 through 9999) of the named status definition record.

If a version number is not specified, VERSION defaults to the system default version number, as specified in the OOAK record at system generation.

If no system default version is specified in the OOAK record, VERSION defaults to 1.

The equals sign (=) can be used in place of IS.

AUTO DISPlay is Yes/No

Specifies whether the first page of pageable map is displayed automatically.

A DISPLAY statement must be coded in the dialog's premap process to display the first page.

YES is the default when neither YES or NO is specified.

PAGing MODE is UPDate/BROwse

Specifies whether the user can modify data fields on a map during a map paging session.

►► For more information, see Chapter 17, “Map Commands.”

UPDATE is the default setting for the paging mode option.

BACKpage is Yes/No

Specifies whether the user can page backward in a map paging session.

►► For more information, see Chapter 17, “Map Commands.”

YES is the default setting for the backpage option.

PAGing TYPE is Nowait/Return/Wait

Specifies the method used to determine the runtime flow of control when the user presses a control key during a map paging session.

►► For more information, see Chapter 17, “Map Commands.”

NOWAIT is the default setting for the paging type option.

These three paging session dialog options can be specified only if the dialog is associated with a pageable map.

The following combination of paging session dialog options cannot be specified: PAGING MODE IS UPDATE, BACKPAGE IS NO, and PAGING TYPE IS NOWAIT.

ACTivity log is Yes/No

Specifies whether the dialog uses the activity logging facility.

This facility documents all potential database activity by a dialog, based on the database commands issued explicitly or implicitly by the dialog's processes.

►► For more information on activity logging, see Appendix E, “Activity Logging for a CA-ADS Dialog.”

The default setting for the activity logging option is defined at DC/UCF system generation.

SYMBOL TABLE is Yes/No

Specifies whether a symbol table is created for a dialog.

A symbol table facilitates the use of element names and process line numbers by the online debugger.

►► For more information on the online debugger:

- See Appendix H, “Debugging a CA-ADS Dialog”
- Refer to *CA-IDMS Online Debugger*

NO is the default setting for the symbol table option.

DIAGnostic TABLE is Yes/No

Specifies whether the dialog load module contains diagnostic tables (line number tables and offset tables).

Diagnostic tables facilitate the testing and debugging of a dialog. If a process aborts, diagnostic tables are used to display the process command in error on the Dialog Abort Information screen. The ADSORPTS utility uses diagnostic tables to format the dialog report for easy reference.

YES is the default setting for the diagnostic table option.

The setting must be YES if the symbol table setting is YES. Also, during the testing of a dialog, the diagnostic table setting should be YES.

Once a dialog has been tested thoroughly, the diagnostic table setting should be NO and the dialog recompiled if dialog load module size is a consideration. The size of a large dialog load module can be reduced significantly by compiling the dialog without diagnostic tables.

MESSAge PREFIX is

Clause introducing a message prefix for a dialog.

The equals sign (=) can be used in place of IS.

prefix

Specifies a user-supplied 2-character alphanumeric message prefix for the dialog.

DEfault

Specifies that the dialog uses the default message prefix.

DEFAULT is the default setting when the message prefix is not specified.

COBoL MOVE is Yes/No

Specifies whether the rules of COBOL or CA-ADS are used in the conversion between data types and in the rounding or truncation of the results of arithmetic and assignment commands.

►► For more information, see:

- Chapter 5, “Introduction to Process Language”
- Chapter 3, “CA-ADS Dialog Compiler (ADSC)”

The default setting for the COBOL MOVE option is defined at DC/UCF system generation. The system generation default is NO.

ENTRY POINT is

Clause introducing the entry point into the dialog when the dialog begins execution at runtime.

EP can be used in place of ENTRY POINT; the equals sign (=) can be used in place of IS.

Premap

Specifies that the dialog begins with its premap process.

PREMAP is the default when no other entry point is specified.

Map

Specifies that the dialog begins with its first mapping operation (mapout for online dialogs, mapin for batch dialogs).

Regardless of the specification, a dialog without an online map or batch input file map begins with its premap process. A dialog without a premap process begins with its first mapping operation.

RETrieval LOCKing is Yes/No

Specifies whether or not the dialog will cause record locks to be held for database records.

YES, the default, specifies that database record retrieval locks will be held on behalf of run units started by the dialog.

►► More information about record retrieval locks can be found in Chapter 16, “Database Access Commands.”

SQL CHEck SYNtax

Specifies the SQL standard you are enforcing. The default is CA-IDMS extended ANSI-standard SQL. CA-ADS supports the following SQL standards:

- EXTended
- SQL89
- FIPS

►► For more information about SQL standards, refer to the *CA-IDMS SQL Reference*.

SQL DATe is

Specifies the external date representation format. The date format can be one of the following:

- ISO specifies the International Standards Organization standard
- USA specifies the IBM USA standard
- EUR specifies the IBM European standard
- JIS specifies the Japanese Industrial Standard Christian Era standard

SQL TIME is

Specifies the external time representation format. The time format can be one of the following:

- ISO specifies the International Standards Organization standard
- USA specifies the IBM USA standard
- EUR specifies the IBM European standard
- JIS specifies the Japanese Industrial Standard Christian Era standard

►► For more information on date/time representations, refer to the *CA-IDMS SQL Reference*.

ADD

Specifies that the SQL table specification is to be added.

MODify

Specifies that the existing SQL table specification is to be replaced by a new SQL table specification.

DELete

Specifies that the SQL table specification is to be cleared to spaces.

SQL TABLE is table-name

Specifies the name of the SQL table assigned the new copy attribute and/or the work record attribute.

table-options

See expansion of table-options below.

ADD

Specifies that a new copy/work record specification is to be added to the dialog.

ADD is the default if the named new copy/work record is not associated with the dialog.

MODify

Specifies that a new copy/work record specification of a dialog is to be modified.

MODIFY is the default if the named new copy/work record is already associated with the dialog.

DELeTe

Specifies that a new copy/work record specification of a dialog is to be deleted.

If the action is DELETE, the VERSION specification is optional, and the NEW COPY and WORK specifications cannot be included.

RECOrd is record-name record-options

Specifies the name of the record assigned the new copy attribute and/or the work record attribute.

See expansion of record-options below.

ADD

Specifies that a premap process is to be added to the dialog.

ADD is the default if no premap process is associated with the dialog.

MODify

Specifies that a new premap process is to replace the existing premap process.

MODIFY is the default if a premap process is already associated with the dialog.

DELeTe

Specifies that the premap process is to be deleted from the dialog.

If the action is DELETE, the version number cannot be specified.

PREmap PROcess NAME is process-name

Specifies the 1- to 32-character name of the process source module associated with the dialog as a premap process.

PROCESS and NAME are optional keywords; the equals sign may be used in place of IS.

Note: The specified process source module must exist in the data dictionary.

VERsion is version-number

Specifies the version number (in the range 1 through 9999) of the named process source module.

The equals sign (=) may be used in place of IS.

The default version number is the system default version number, as specified in the OOAK record at system generation. If no system default version number is specified in the OOAK record, the default version number is 1.

ADD

Specifies that a premap process is to be added to the dialog.

ADD is the default if no premap process is associated with the dialog.

MODify

Specifies that a new premap process is to replace the existing premap process.

MODIFY is the default if a premap process is already associated with the dialog.

DELeTe

Specifies that the premap process is to be deleted from the dialog.

If the action is DELETE, the version number cannot be specified.

DECLaration PROcess NAME is process-name

Specifies the 1- to 32-character name of the process source module associated with the dialog as a premap process.

PROCESS and NAME are optional keywords; the equals sign may be used in place of IS.

Note: The specified process source module must exist in the data dictionary.

VERsion is version-number

Specifies the version number (in the range 1 through 9999) of the named process source module.

The equals sign (=) may be used in place of IS.

The default version number is the system default version number, as specified in the OOAK record at system generation. If no system default version number is specified in the OOAK record, the default version number is 1.

ADD

Specifies that a response process is to be added to the dialog.

ADD is the default if the named response process is not already associated with the dialog.

ADD can be used to define duplicate response processes, in which the same response process is associated with several control keys and/or response field values. In the example shown below, response process RP1 is associated with control keys PF1, PF2, and PF3, and with response field values ADD and MOD:

```
ADD RESPONSE PROCESS RP1 CONTROL KEY PF1 RES VALUE ADD
ADD RESPONSE PROCESS RP1 CONTROL KEY PF2 RES VALUE MOD
ADD RESPONSE PROCESS RP1 CONTROL KEY PF3
```

MODify

Specifies that a response process of the dialog is to be modified.

MODIFY is the default if the named response process is already associated with the dialog.

In the example shown below, the control key specification for nonduplicate response process RP1 is modified to PF2:

```
MODIFY RESPONSE PROCESS RP1 CONTROL KEY PF2
```

To modify a duplicate response process, the application developer must specify which occurrence of the duplicate response process is being modified.

To modify the control key associated with the response process, the application developer specifies the FROM parameter of the CONTROL KEY specification. In the example shown below, the control key ENTER is changed to PA1 for duplicate response process RP1:

```
MODIFY RESPONSE PROCESS RP1 CONTROL KEY PA1 FROM ENTER
```

To modify the response field value associated with the response process, the application developer specifies the FROM parameter of the RESPONSE FIELD VALUE specification. In the example shown below, the response field value MOD is changed to ADD for duplicate response process RP1:

```
MODIFY RESPONSE PROCESS RP1 RES VALUE ADD FROM MOD
```

To modify the EXECUTE ON EDIT ERRORS specification associated with the response process, the application developer specifies either the CONTROL KEY or RESPONSE FIELD VALUE parameter. In the example shown below, the EXECUTE ON EDIT ERRORS specification is set to YES for the occurrence of duplicate response process RP1 that is associated with the ENTER key:

```
MODIFY RESPONSE PROCESS RP1 CONTROL KEY ENTER  
EXECUTE ON EDIT ERRORS YES
```

DELete

Specifies that a response process of the dialog is to be deleted.

If the action is DELETE, the version number is optional and the EXEC ON EDIT ERRORS specifications cannot be included.

An occurrence of a duplicate response process is deleted by specifying the CONTROL KEY or RESPONSE FIELD VALUE parameter. The example shown below deletes the occurrence of duplicate response process RP1 that is associated with the response field value ADD:

```
DELETE RESPONSE PROCESS RP1 RES VALUE ADD
```

RESponse response-options

See expansion of response-options below.

Expansion of response-options**RESponse PROcess NAME**

Specifies the 1- to 32-character name of the process source module associated with the dialog as a response process.

Note: The specified source module must exist in the data dictionary.

VERsion is version-number

Specifies the version number (in the range 1 through 9999) of the named process source module.

The default version number is the system default version number, as specified in the OOK record at system generation. If no system default version number is specified in the OOK record, the default version number is 1.

The equals sign (=) may be used in place of IS.

DEFault is Yes/No

Specifies whether the response process defined is the optional default response process of the dialog.

At runtime, after a mapin operation, the runtime system executes the default response process if no response process can be selected based on control event or response field values.

NO is the default specification.

If DEFAULT is NO, a control key, a response field value, a response field value, or a batch control event for the response process must be specified. If DEFAULT is YES, these specifications are optional.

CONTROL KEY is key

Specifies a user-defined control key that initiates the response process at runtime.

The equals sign (=) can be used in place of IS.

Key can also be specified to identify an occurrence of a duplicate response process, as described under ADD/MODIFY/DELETE RESPONSE PROCESS above.

Valid control key specifications are ENTER, CLEAR, PA1 through PA3, PF1 through PF24, FWD, BWD, and HDR. FWD, BWD, and HDR can be specified only if the dialog is associated with a pageable map. LPEN can be specified as a control key if the use of light pens is supported by the installation.

CLEAR, PA1, PA2, and PA3 do not transmit data; that is, input is not mapped in when these keys are pressed at runtime. The FWD, BWD, and HDR control keys are associated with pageable maps. FWD and BWD are synonymous with the keyboard control keys for paging forward and backward, respectively. If FWD and BWD are specified and the keys defined for paging forward and backward are changed, the dialog does not have to be recompiled.

HDR is not associated with any keyboard control key; rather, conditions encountered during a map paging session cause a response process associated with this control key value to be initiated.

►► For more information on the effect of HDR on the runtime flow of control, see in Chapter 4, “CA-ADS Runtime System.”

FROM old-key

Identifies the occurrence of a duplicate response process whose associated control key specification is being modified, as described under ADD/MODIFY/DELETE RESPONSE PROCESS above.

RESponse FIEld VALue is value

Specifies a response name associated with the response process.

Value can also be specified to identify an occurrence of a duplicate response process, as described under ADD/MODIFY/DELETE RESPONSE PROCESS above.

The equals sign (=) may be used in place of IS.

When a control key value or a response field value of a response process needs to be dissociated from the response, a blank value (' ') can be used, as in the following example:

```
MOD RES PRO response-name VER 1 RES VALUE ' '
```

FROM old-value

Identifies the occurrence of a duplicate response process whose associated response field value specification is being modified, as described under ADD/MODIFY/DELETE RESPONSE PROCESS above.

BATch CONTROL EVENT is event

Specifies a batch control event that initiates the response process at runtime.

BCE can be used in place of BATCH CONTROL EVENT; the equals sign (=) can be used in place of IS.

Valid batch control events

- **EOF** indicates that the most recent input file read operation resulted in an end-of-file condition.
- **IOERR** indicates that the most recent input file read operation resulted in physical input-error condition. In CA-ADS/Batch, output errors cause the runtime system to terminate the application.

Batch control events can be specified only for batch dialogs. Control keys can be specified only for online dialogs.

EXEc ON EDIt ERRors is

Introduces whether processing continues if automatic editing encounters map input errors.

The equals sign (=) can be used in place of IS.

Yes

Specifies that the response process executes even if the map contains input errors.

No

Specifies that the response process is not executed if the map contains input errors. The user must correct all map fields that are in error before processing continues.

NO is the default when neither YES or NO is specified.

Expansion of table-options

SCHema is sql-schema-name

Specifies the schema containing the SQL table.

The equals sign (=) can be used in place of IS.

NEW copy

Specifies that the table is assigned the new copy attribute.

Records with the new copy attribute are allocated new table buffers when the dialog executes at runtime.

NC can be used in place of NEW COPY.

WORK

Specifies that the table is assigned the work attribute.

Records with the work table attribute are available to the dialog as working storage at runtime.

WK can be used in place of WORK.

If no attribute is specified for the named table, WORK is assigned as the default.

If NEW COPY is specified for the table, WORK is not automatically assigned; the application developer must explicitly specify the work table attribute.

►► For more information on the new copy and work attributes, see Chapter 3, "CA-ADS Dialog Compiler (ADSC)."

Expansion of record-options

VERsion is version-number

Specifies the version number (in the range 1 through 9999) of the named record.

If a version number is not specified, VERSION defaults to the system default version number, as specified in the OOAK record at system generation.

If no system default version number is specified in the OOAK record, VERSION defaults to 1.

The equals sign (=) can be used in place of IS.

NEW copy

Specifies that the record is assigned the new copy attribute.

Records with the new copy attribute are allocated new record buffers when the dialog executes at runtime.

NC can be used in place of NEW COPY.

WORK

Specifies that the record is assigned the work attribute.

Records with the work record attribute are available to the dialog as working storage at runtime.

WK can be used in place of WORK.

If no attribute is specified for the named record, WORK is assigned as the default. If NEW COPY is specified for the record, WORK is not automatically assigned; the application developer must explicitly specify the work record attribute.

►► For more information on the new copy and work attributes, see Chapter 3, “CA-ADS Dialog Compiler (ADSC).”

Expansion of version#-options

version-number

Specifies a single version number for the selected dialogs.

low-version-number high-version-number

Specifies all versions of the selected dialogs within the version-number range (inclusive).

The default version number is 1.

Usage:

Considerations

- The DIALOG clause must be the first clause of a dialog expression.
- The MAINLINE, SUBSCHEMA, MAPNAME, AUTOSTATUS, STATUS DEFINITION RECORD, ACTIVITY LOG, SYMBOL TABLE, DIAGNOSTIC TABLE, MESSAGE PREFIX, COBOL MOVE, ENTRY POINT, and RETRIEVAL LOCKING clauses can appear in any order, but must precede the first RECORD clause.
- The PAGING MODE, BACKPAGE, and PAGING TYPE clauses can be specified only if the dialog's map is pageable. These clauses can appear in any order, but must follow the MAPNAME clause and precede the first RECORD clause.
- All RECORD clauses must precede the first PROCESS clause.
- PREMAP PROCESS and RESPONSE PROCESS clauses can appear in any order, provided that the above requirements are met.

Examples: Example 1: Recompiling all dialogs

All dialogs in the load area are recompiled:

```
COMPILE FROM LOAD ALL.
```

Example 2: Recompiling dialogs by version number

All dialogs with version number 2 and version numbers 5 through 8 are recompiled:

```
COMPILE FROM LOAD ALL VERSION ( 2 ( 5 8 ) ).
```

Example 3: Recompiling dialogs by name

All dialogs with names that begin with C and that have the letters D and R in the fourth and fifth positions are recompiled:

```
COMPILE FROM LOAD DIALOG (C**DR***).
```

Example 4: Recompiling dialogs within a specified range

Dialogs QWERT001 through ZZZZZZZZ are recompiled:

```
COMPILE FROM LOAD DIALOG ( ( QWERT001 ZZZZZZZZ ) ).
```

Example 5: Recompiling an added dialog

The dialog SXADIAL is added and compiled:

```
COMPILE FROM SOURCE
  ADD DIALOG SXADIAL VER IS 1 MAINLINE YES
  ADD SUBSCHEMA DEMOSS01 SCHEMA DEMOSCHM VER 1
  ADD MAPNAME SXA1 VER IS 1
  ADD REC CUSTOMER VER 2 NC
  ADD REC SXAREC1 VER 1 NC WK
  ADD PREMAP SXAPREMAP VER 1
  ADD RESPONSE PROCESS NAME SXARESP5 VER 2 CONTROL KEY
    PF5 EXEC NO
  ADD RESPONSE PROCESS NAME SXARESP3 VER 1 CONTROL KEY ENTER
    RESPONSE FIELD SXARESP4 EXEC NO.
```

D.2.7 JCL and commands

JCL and commands for running ADSOBCOM are shown below for OS/390, VSE/ESA, VM/ESA, and BS2000/OSD systems.

D.2.7.1 OS/390 JCL

Sample OS/390 JCL for central version: ADSOBCOM (OS/390)

```
//          EXEC PGM=ADSOBCOM,REGION=500K
//STEPLIB  DD  DSN=idms.loadlib,DISP=SHR
//CDMSLIB  DD  DSN=idms.loadlib,DISP=SHR
//sysctl   DD  DSN=idms.sysctl,DISP=SHR
//dclscr   DD  DSN=cdms.dclscr,DISP=SHR
//SYSLST   DD  SYSOUT=A
//SYSUDUMP DD  SYSOUT=A
//SYSIDMS  DD  *
SYSIDMS parameters
//SYSIPT   DD  *
control statements
/*
```

<u>idms.loadlib</u>	data set name of the CA-IDMS load library
<u>idms.sysctl</u>	data set name of the SYSCTL file
<u>dclscr</u>	ddname of the local scratch file (if one is specified in the DMCL; otherwise not required)
<u>cdms.dclscr</u>	data set name of the local scratch file (if one is specified; otherwise not required)
<u>sysctl</u>	ddname of the SYSCTL file
<u>SYSIDMS parameters</u>	a list of the SYSIDMS parameters that pertain to this job

Note: The CDMSLIB must contain the CA_C runtime library components.

The local scratch file need not be specified if the SYSIDMS parameterXA_SCRATCH=ON is specified.

►► See *CA-IDMS Database Administration*, for more information about SYSIDMS parameters.

Sample OS/390 JCL for local mode: To execute ADSOBCOM in **local mode**, perform the following steps:

1. Remove the sysctl DD statement.
2. Add the following statements after the CDMSLIB DD statement:

```
//sysjrn1 DD DSN=idms.tapejrn1,DISP=(NEW,KEEP),UNIT=tape
//dictdb  DD DSN=idms.appldict.ddldml,DISP=SHR
//dloddb  DD DSN=idms.appldict.ddldclod,DISP=SHR
//dmsgdb  DD DSN=idms.sysmsg.ddldcmsg,DISP=SHR
```

<u>idms.appldict.ddldml</u>	data set name of the data dictionary DDLDDL area
<u>idms.appldict.ddldclod</u>	data set name of the data dictionary load area
<u>idms.sysmsg.ddldcmsg</u>	data set name of the data dictionary message area
<u>idms.tapejrn1</u>	data set name of the tape journal file
<u>dictdb</u>	ddname of the data dictionary DDLDDL area
<u>dloddb</u>	ddname of the data dictionary load area (DDLDCLOD)
<u>dmsgdb</u>	ddname of the data dictionary message area (DDLDM/ESAG)
<u>sysjrn1</u>	ddname of the tape journal file
<u>tape</u>	symbolic device name of the tape journal file

D.2.7.2 VSE/ESA JCL

Sample VSE/ESA JCL for central version: ADSOBCOM (VSE/ESA)

```
// UPSI b          if specified in ADS00PTI module
// DLBL      userlib
// EXTENT    ,nnnnnn
// LIBDEF    *,SEARCH=(userlib.cdmslib)
// DLBL      dclscr,, 'cdms.dclscr',,DA
// EXTENT    sys014,nnnnnn
// ASSGN     sys014,DISK,VOL=nnnnnn,SHR
// EXEC      ADSOBCOM
control statements
SYSIDMS parameters
```

<u>b</u>	appropriate 1- to 8-character UPSI bit switch, as specified in the IDMSOPTI module
<u>cdms.dclscr</u>	file-id of the local scratch area
<u>dclscr</u>	filename of the local scratch area (if one is specified in the DMCL, otherwise not required)
<u>sys014</u>	logical unit assignment for the local scratch area (if one is specified in the DMCL, otherwise not required)
<u>nnnnnn</u>	volume serial number of the library
<u>userlib</u>	filename of the CA-IDMS library
<u>userlib.cdmslib</u>	file-id of the CA-IDMS sublibrary
<u>SYSIDMS parameters</u>	A list of SYSIDMS parameters for this job

►► See *CA-IDMS Database Administration*, for more information about SYSIDMS parameters.

Sample VSE/ESA JCL for local mode: To execute ADSOBCOM in **local mode**, perform the following steps:

1. Remove the UPSI specification.
2. Add the following statements before the EXEC statement:

```
// DLBL      dictdb,'idms.appldict.ddldm1',,DA
// EXTENT    sys015,nnnnnn
// ASSGN     sys015,DISK,VOL=nnnnnn,SHR
// DLBL      dloddb,'idms.appldict.ddldclod',,DA
// EXTENT    sys017,nnnnnn
// ASSGN     sys017,DISK,VOL=nnnnnn,SHR
// DLBL      dmsgdb,'idms.sysmsg.ddldcmsg',,DA
// EXTENT    sys016,nnnnnn
// ASSGN     sys016,DISK,VOL=nnnnnn,SHR
// TLBL      sys009,'idms.tapejrn1',,nnnnnn,,f
// ASSGN     sys009,TAPE,VOL=nnnnnn
```

<u>idms.appldict.ddldml</u>	file-id of the data dictionary DDLDDL area
<u>idms.appldict.ddldclod</u>	file-id of the data dictionary load area
<u>idms.sysmsg.ddldcmsg</u>	file-id of the data dictionary message area
<u>idms.tapejrn1</u>	file-id of the tape journal file
<u>dictdb</u>	filename of the data dictionary DDLDDL area
<u>dloddb</u>	filename of the data dictionary load area (DDLDCLOD)
<u>dmsgdb</u>	filename of the data dictionary message area (DDLDM/ESAG)
<u>f</u>	file number of the tape journal file
<u>nnnnnn</u>	volume serial number
<u>sys009</u>	logical unit assignment for the tape journal file
<u>sys015</u>	logical unit assignment for the data dictionary DDLDDL area
<u>sys016</u>	logical unit assignment for the data dictionary message area
<u>sys017</u>	logical unit assignment for the data dictionary load area

D.2.7.3 VM/ESA commands

Sample VM/ESA commands for central version: ADSOBCOM (VM/ESA)

```
FILEDEF SYSLST PRINTER
FILEDEF SYSIDMS DISK sysidms input a
FILEDEF SYSIPT DISK bgen input a
GLOBAL LOADLIB idmslib
OSRUN ADSOBCOM
```

<u>sysidms input a</u>	filename, filetype, and filemode of the file containing the SYSIDMS input parameters
<u>bgen input a</u>	file identifier of the file containing ADSOBCOM source statements
<u>idmslib</u>	filename of the CA-IDMS LOADLIB library

►► See *CA-IDMS Database Administration*, for more information about SYSIDMS parameters.

Sample VM/ESA commands for local mode: To execute ADSOBCOM in **local mode**, add the following commands before the OSRUN command:

```
FILEDEF sysjrn1 TAP1 SL VOLID nnnnnn (RECFM VB LRECL 111 BLKSIZE bbb
FILEDEF dictdb DISK dictdb dictfile d (RECFM F LRECL ppp BLKSIZE ppp XTENT nnn
FILEDEF dloddb DISK dloddb dictfile f (RECFM F LRECL ppp BLKSIZE ppp XTENT nnn
FILEDEF dmsgdb DISK dmsgdb dictfile e (RECFM F LRECL ppp BLKSIZE ppp XTENT nnn
```

<u>bbb</u>	block size of the tape journal file
<u>dictdb</u>	ddname of the data dictionary DDLDDL area
<u>dictdb dictfile d</u>	file identifier of the data dictionary DDLDDL area
<u>dloddb</u>	ddname of the data dictionary load area (DDLDCLOD)
<u>dloddb dictfile f</u>	file identifier of the data dictionary load area
<u>dmsgdb</u>	ddname of the data dictionary message area (DDLDM/ESAG)
<u>dmsgdb dictfile e</u>	file identifier of the data dictionary message area
<u>111</u>	record length of the tape journal file
<u>nnnnnn</u>	volume serial number of the tape journal file
<u>ppp</u>	page size of the area
<u>sysjrn1</u>	ddname of the tape journal file

Specifying central version or local mode: To specify whether ADSOBCOM executes under central version or in local mode, take one of the following actions:

1. Specify either CVMACH=*dc/ucf-machine-name* (for central version) or *LOCAL* (for local mode) as the first statement submitted to ADSOBCOM.
Dc/ucf-machine-name is the 1- through 8-character user identifier of the VM/ESA virtual machine in which the DC/UCF system is executing.
2. Link edit ADSOBCOM with an IDMSOPTI module that specifies either CVMACH=*dc/ucf-machine-name* (for central version) or CENTRAL=NO (for local mode). Instructions for creating an IDMSOPTI module are given in *CA-IDMS System Operations*.
3. Code PARM='CVMACH=*dc/ucf-machine-name*' or PARM='*LOCAL*' on the OSRUN command used to invoke the compiler. This option is not allowed if the OSRUN command is issued from a VM/ESA EXEC program; however, it is allowed if the OSRUN command is issued from a System Product interpreter (REXX) or EXEC 2 program.

Additional information about central version and local mode operations in the VM/ESA environment can be found in *Installation and Maintenance Guide — VM/ESA*.

D.2.7.4 BS2000/OSD JCL

Sample BS2000/OSD JCL for central version: ADSOBCOM (BS2000/OSD)

```
/ADD-FILE-LINK L-NAME=CDMSLIB,F-NAME=idms.dba.loadlib
/ADD-FILE-LINK L-NAME=CDMSLIB1,F-NAME=idms.loadlib
/ADD-FILE-LINK L-NAME=CDMSLODR,F-NAME=idms.loadlib
/ADD-FILE-LINK L-NAME=sysctl,F-NAME=idms.sysctl,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=SYSIDMS,F-NAME=idms.sysidms
/ASSIGN-SYSDTA TO=*SYSCMD
/START-PROG *MOD(ELEM=ADSOBCOM,LIB=idms.loadlib,RUN-MODE=*ADV)
control statements
```

<u>idms.loadlib</u>	filename of the CA-IDMS system load library
<u>idms.dba.loadlib</u>	data set name of the load library containing the DMCL and database name table load modules
<u>idms.sysidms</u>	filename of the file containing the SYSIDMS parameters
<u>idms.sysctl</u>	filename of the SYSCTL file
<u>sysctl</u>	linkname of the SYSCTL file

►► See *CA-IDMS Database Administration*, for more information about SYSIDMS parameters.

Sample BS2000/OSD JCL for local mode: To execute ADSOBCOM in **local mode**, perform the following steps:

1. Remove the ADD-FILE-LINK statement for the sysctl
2. Add the following statements:

```
/ADD-FILE-LINK L-NAME=dictdb,F-NAME=idms.appldict.dictdb,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=d1oddb,F-NAME=idms.appldict.d1oddb,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=dcmmsg,F-NAME=idms.sysmsg.dd1dcmmsg,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=sysjrn1,F-NAME=idms.tapejrn1
```

<u>idms.appldict.ddldml</u>	filename of the data dictionary DDLML area
<u>idms.appldict.ddldclod</u>	filename of the data dictionary load area
<u>idms.sysmsg.ddldcmsg</u>	filename of the data dictionary message area
<u>idms.tapejrn1</u>	filename of the tape journal file
<u>dictdb</u>	linkname of the data dictionary DDLML area
<u>dloddb</u>	linkname of the data dictionary load area (DDLDCLOD)
<u>dmsgdb</u>	linkname of the data dictionary message area (DDLDM/ESAG)
<u>sysjrn1</u>	linkname of the tape journal file

D.3 ADSOBSYS

The ADSOBSYS utility builds a load module (ADSOOPTI) that supplies CA-ADS system generation parameters to ADSOBCOM. ADSOBSYS must be run once for each DC/UCF system at installation and whenever CA-ADS system generation parameters are changed.

The ADSOOPTI module can be either loaded at runtime by ADSOBCOM or link edited with ADSOBCOM. Note that with dynamic loading, the module must have the default ADSOOPTI module name.

ADSOBSYS can also supply system generation parameters to the CA-ADS/Batch runtime system.

ADSOBSYS uses standard control statements in addition to the SYSTEM statement. The control statements and the JCL used to run ADSOBSYS are presented below. Parameters given for the SYSTEM statement apply to CA-ADS applications.

D.3.1 Control statements

The following control statements can be used with ADSOBSYS:

ICTL: Specifies scanning a specified column range for meaningful data. The default specification is 1-72. The ICTL statement format is shown below:

```
➤ ┌ ICTL = (start-column-number end-column-number) ─┐ ➤
```

OCTL: Specifies the number of lines to appear on each page of the ADSOBSYS printed output. The default specification is 56. The OCTL statement format is shown below:

```
➤ ┌ OCTL = (line-count-number) ─┐ ➤
```

ISEQ: Specifies sequence checking on source statements falling within a specified column range. The ISEQ statement format is shown below:

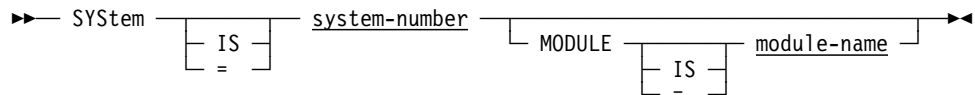
```
➤ ┌ ISEQ = (start-column-number end-column-number) ─┐ ➤
```

The ICTL, OCTL, and ISEQ control statements must be submitted to the job stream before the SYSTEM statement.

D.3.2 SYSTEM statement

Purpose: Specifies the DC/UCF system for which the ADSOOPTI module is being created.

Syntax:



Parameters

SYStem IS system-number

Specifies the 1- to 4-digit number of the DC/UCF system for which the ADSOOPTI module is being created.

MODULE IS module-name

Specifies the 1 to 8-character name of the module being created. The default module name is ADSOOPTI.

D.3.3 JCL and commands

JCL and commands for running ADSOBSYS are shown below for OS/390, VSE/ESA, VM/ESA, and BS2000/OSD systems.

D.3.3.1 OS/390 JCL

**Sample OS/390 JCL for central version: ADSOBSYS (central version)
(OS/390)**

```
//ADSOBSYS EXEC PGM=ADSOBSYS,REGION=1024K
//STEPLIB DD DSN=idms.dba.loadlib,DISP=SHR
// DD DSN=idms.loadlib,DISP=SHR
//sysctl DD DSN=idms.sysctl,DISP=SHR
//dcmmsg DD DSN=idms.sysmsg.ddldcmmsg,DISP=SHR
//SYSLST DD SYSOUT=A
//SYSPCH DD DSN=&&object,DISP=(NEW,PASS),
// UNIT=disk,SPACE=(TRK,1),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSIDMS DD *
DMCL=dmc1-name
DBNAME=system
Put other SYSIDMS parameters, as appropriate, here
/*
//SYSIPT DD *
Put ADSOBSYS parameters, as appropriate, here
/*
/*
//LINKOPTI EXEC PGM=IEWL,REGION=1024K,PARM='LET,LIST,NCAL,XREF'
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD UNIT=disk,SPACE=(CYL,(3,2))
//SYSLMOD DD DSN=idms.dba.loadlib,DISP=SHR
//SYSLIN DD DSN=&&object,DISP=(OLD,DELETE)
// DD *
ENTRY adsoopti
NAME adsoopti(R)
/*
/*
```

Sample OS/390 JCL for local mode ADSOBSYS (local mode) (OS/390)

```
//ADSOBSYS EXEC PGM=ADSOBSYS,REGION=1024K
//STEPLIB DD DSN=idms.dba.loadlib,DISP=SHR
// DD DSN=idms.loadlib,DISP=SHR
//dcdm1 DD DSN=idms.system.ddldm1,DISP=SHR
//dcmsg DD DSN=idms.sysmsg.ddldcmsg,DISP=SHR
//sysjrn1 DD DUMMY
//SYSLST DD SYSOUT=A
//SYSPCH DD DSN=&&object,DISP=(NEW,PASS),
// UNIT=disk,SPACE=(TRK,1),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSIDMS DD *
DMCL=dmc1-name
DBNAME=system
Put other SYSIDMS parameters, as appropriate, here
/*
//SYSIPT DD *
Put ADSOBSYS parameters, as appropriate, here
/*
/*
//LINKOPTI EXEC PGM=IEWL,REGION=1024K,PARM='LET,LIST,NCAL,XREF'
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD UNIT=disk,SPACE=(CYL,(3,2))
//SYSLMOD DD DSN=idms.dba.loadlib,DISP=SHR
//SYSLIN DD DSN=&&object,DISP=(OLD,DELETE)
// DD *
ENTRY adsoopti
NAME adsoopti(R)
/*
/*
```

<u>idms.dba.loadlib</u>	Data set name of the load library containing the DMCL and database name table load modules
<u>idms.loadlib</u>	Data set name of the load library containing the CA-IDMS executable modules
<u>sysctl</u>	DDname of the SYSCTL file
<u>idms.sysctl</u>	Data set name of the SYSCTL file
<u>dcmsg</u>	DDname of the system message (DDLDM/ESAG) area
<u>idms.sysmsg.ddldcmsg</u>	Data set name of the system message (DDLDM/ESAG) area
<u>&&object</u>	Temporary data set name for the ADSOOPTI object module
<u>disk</u>	Symbolic device name for the work files
<u>dmcl-name</u>	Name of the DMCL load module
<u>system</u>	Name of the system dictionary
<u>adsoopti</u>	ADSOOPTI module name
<u>dcddl</u>	DDname of the system dictionary definition (DDLDM) area
<u>idms.system.ddldml</u>	Data set name of the system dictionary definition (DDLDM) area
<u>sysjrn1</u>	DDname of the tape journal file

►► See *CA-IDMS Database Administration*, for more information about SYSIDMS parameters.

D.3.3.2 VSE/ESA JCL

Sample VSE/ESA JCL for central version: ADSOBSYS (VSE/ESA)

```
// UPSI b                if specified in IDMSOPTI module
// DLBL      userlib
// EXTENT    ,nnnnnn
// LIBDEF    *,SEARCH=(userlib.cdmslib)
// LIBDEF    *,CATALOG=(userlib.cdmslib)
// DLBL      IDMSPCH,'temp.adsoopti'
// EXTENT    sysnnn,nnnnnn,,ssss,1111
// ASSGN     sysnnn,DISK,VOL=nnnnnn,SHR
// EXEC      ADSOBSYS
//           SYSTEM=nnnn,MODULE=adsoopti
SYSIDMS parameters
/*
// DLBL      IJSYSIN,'temp.adsoopti'
// EXTENT    SYSIPT,nnnnnn
// ASSGN     SYSIPT,DISK,VOL=nnnnnn,SHR
// OPTION    CATAL
// PHASE     adsoopti,*
// INCLUDE
// ENTRY     (adsoopti)
// EXEC      LNKEDT
// CLOSE     SYSIPT,SYSRDR
// CLOSE     sysc1b,UA
```

<u>adsoopti</u>	ADSOOPTI module name
<u>SYSIDMS parameters</u>	a list of SYSIDMS parameters for this job
<u>b</u>	appropriate 1- to 8-character UPSI bit switch, as specified in the IDMSOPTI module
<u>1111</u>	number of tracks (CKD) or blocks (FBA) of the disk extent
<u>nnnn</u>	version number of the DC/UCF system
<u>nnnnnn</u>	volume serial number of the library
<u>ssss</u>	starting track (CKD) or block (FBA) of the disk extent
<u>sysnnn</u>	logical unit assignment of the temporary adsoopti module
<u>temp.adsoopti</u>	temporary file-id of the ADSOOPTI module
<u>userlib</u>	filename of the user library
<u>userlib.cdmslib</u>	file-id of the CA-IDMS sublibrary

►► See *CA-IDMS Database Administration*, for more information about SYSIDMS parameters.

Sample VSE/ESA JCL for local mode: To execute ADSOBSYS in **local mode**, perform the following steps:

1. Remove the UPSI specification.
2. Add the following statements before the EXEC ADSOBSYS statement:

```
// DLBL      dictcb,'idms.appldict.ddldml',,DA
// EXTENT    sys015,nnnnnn
// ASSGN      sys015,DISK,VOL=nnnnnn,SHR
// DLBL      dloddb,'idms.appldict.ddldclod',,DA
// EXTENT     sys017,nnnnnn
// ASSGN      sys017,DISK,VOL=nnnnnn,SHR
// DLBL      dmsgdb,'idms.sysmsg.ddldcmsg',,DA
// EXTENT     sys016,nnnnnn
// ASSGN      sys016,DISK,VOL=nnnnnn,SHR
// TLBL      sys009,'idms.tapejrn1',, nnnnnn,,f
// ASSGN      sys009,TAPE,VOL=nnnnnn
```

<u>idms.appldict.ddldml</u>	file-id of the data dictionary DDLML area
<u>idms.appldict.ddldclod</u>	file-id of the data dictionary load area
<u>idms.sysmsg.ddldcmsg</u>	file-id of the data dictionary message area
<u>idms.tapejrn1</u>	file-id of the tape journal file
<u>dictdb</u>	filename of the data dictionary DDLML area
<u>dloddb</u>	filename of the data dictionary load area (DDLDCLOD)
<u>dmsgdb</u>	filename of the data dictionary message area (DDLDM/ESAG)
<u>f</u>	file number of the tape journal file
<u>nnnnnn</u>	volume serial number of the library
<u>sys009</u>	logical unit assignment for the tape journal file
<u>sys015</u>	logical unit assignment for the data dictionary DDLML area
<u>sys016</u>	logical unit assignment for the data dictionary message area
<u>sys017</u>	logical unit assignment for the data dictionary load area
<u>userlib</u>	filename of the user library
<u>userlib.cdmslib</u>	file-id of the CA-IDMS sublibrary

D.3.3.3 VM/ESA commands

Sample VM/ESA commands for central version: ADSOBSYS (VM/ESA)

```
FILEDEF SYSLST PRINTER
FILEDEF SYSPCH DISK opti TEXT a (LRECL 80 BLKSIZE 400 RECFM FB
FILEDEF SYSIDMS DISK sysidms input a
FILEDEF SYSIPT DISK bsys input a
GLOBAL LOADLIB idmslib
OSRUN ADSOBSYS
FILEDEF SYSPRINT PRINTER
TXTLIB DEL utextlib opti
TXTLIB ADD utextlib opti
FILEDEF SYSLMOD DISK uloadlib LOADLIB a6 (RECFM V LRECL 1024 BLKSIZE 1024
FILEDEF objlib DISK utextlib TXTLIB a
LKED linkctl (LET LIST NCAL
```

Linkage editor control statements (linkctl):

```
INCLUDE objlib (opti)
ENTRY opti
NAME opti(R)
```

<u>sysidms input a</u>	filename, filetype, and filemode of the file containing the SYSIDMS input parameters
<u>bsys input a</u>	file identifier of the file containing ADSOBSYS source statements
<u>idmslib</u>	filename of the CA-IDMS LOADLIB library
<u>linkctl</u>	filename of the file containing the linkage editor control statements; the file must have the filetype of TEXT
<u>objlib</u>	ddname of the user TXTLIB library
<u>opti</u>	filename of the file for the ADSOOPTI module
<u>opti TEXT a</u>	file identifier of the file for the ADSOOPTI module
<u>uloadlib LOADLIB a6</u>	file identifier of the user LOADLIB library
<u>utextlib</u>	filename of a user TXTLIB library

►► See *CA-IDMS Database Administration*, for more information about SYSIDMS parameters.

Sample VM/ESA commands for local mode: To execute ADSOBSYS in **local mode**, add the following commands before the OSRUN command:

```
FILEDEF sysjrn1 TAP1 SL VOLID nnnnnn (RECFM VB LRECL 111 BLKSIZE bbb
FILEDEF dictdb DISK dictdb dictfile d (RECFM F LRECL ppp BLKSIZE ppp XTENT nnn
FILEDEF dloddb DISK dloddb dictfile f (RECFM F LRECL ppp BLKSIZE ppp XTENT nnn
FILEDEF dmsgdb DISK dmsgdb dictfile e (RECFM F LRECL ppp BLKSIZE ppp XTENT nnn
```

<u>bbb</u>	block size of the tape journal file
<u>dictdb</u>	ddname of the data dictionary DDLDDL area
<u>dictdb dictfile d</u>	file identifier of the data dictionary DDLDDL area
<u>dloddb</u>	ddname of data dictionary load area (DDLDCLOD)
<u>dloddb dictfile f</u>	file identifier of the data dictionary load area
<u>dmsgdb</u>	ddname of data dictionary message area (DDLDM/ESAG)
<u>dmsgdb dictfile e</u>	file identifier of the data dictionary message area
<u>l11</u>	record length of the tape journal file
<u>nnnnnn</u>	volume serial number of the tape journal file
<u>ppp</u>	page size of the area
<u>sysjrn1</u>	ddname of the tape journal file

Specifying central version or local mode: To specify whether ADSOBSYS executes under central version or in local mode, perform one of the following actions:

1. Specify either CVMACH=*dc/ucf-machine-name* (for central version) or *LOCAL* (for local mode) as the first statement to submit to ADSOBSYS. *Dc/ucf-machine-name* is the 1- through 8-character user identifier of the VM/ESA virtual machine in which the DC/UCF system is executing.
2. Link edit ADSOBSYS with an IDMSOPTI module that specifies either CVMACH=*dc/ucf-machine-name* (for central version) or CENTRAL=NO (for local mode). Instructions for creating an IDMSOPTI module are given in *CA-IDMS System Operations*.
3. Code PARM='CVMACH=*dc/ucf-machine-name*' or PARM='*LOCAL*' on the OSRUN command used to invoke the compiler. This option is not allowed if the OSRUN command is issued from a VM/ESA EXEC program; however, it is allowed if the OSRUN command is issued from a System Product interpreter (REXX) or EXEC 2 program.

Additional information about central version and local mode operations in the VM/ESA environment can be found in *CA-IDMS Installation and Maintenance Guide* — *VM/ESA*.

D.3.3.4 BS2000/OSD JCL

Sample BS2000/OSD JCL for central version: ADSOBSYS (BS2000/OSD)

```

/ADD-FILE-LINK L-NAME=CDMSLIB,F-NAME=idms.dba.loadlib
/ADD-FILE-LINK L-NAME=CDMSLIB1,F-NAME=idms.loadlib
/ADD-FILE-LINK L-NAME=CDMSLODR,F-NAME=idms.loadlib
/ADD-FILE-LINK L-NAME=sysctl,F-NAME=idms.sysctl,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=SYSIDMS,F-NAME=idms.sysidms
/ASSIGN-SYSOPT TO=temp.pch
/ASSIGN-SYSDTA TO=*SYSCMD
/START-PROG *MOD(ELEM=ADSOBSYS,LIB=idms.loadlib,RUN-MODE=*ADV)
SYSTEM=nnnn,MODULE=adsoopti
/ASSIGN-SYSOPT TO=*PRIMARY
/ADD-FILE-LINK L-NAME=OBJMINPT,F-NAME=temp.pch
/ADD-FILE-LINK L-NAME=OBJMLIB0,F-NAME=idms.objlib.user
/START-PROG *MOD(ELEM=BS2K0BJM,LIB=idms.loadlib,RUN-MODE=*ADV)
MODULE=adsoopti
/START-BINDER
//START-LLM-CREATION INTERNAL-NAME=adsoopti
//INC-MOD LIB=idms.objlib.user,ELEM=adsoopti
//SAVE-LLM LIB=idms.dba.loadlib,ELEM=adsoopti(VER=0),OVER=YES
//END

```

<u>adsoopti</u>	ADSOOPTI module name
<u>idms.loadlib</u>	filename of the CA-IDMS load library
<u>idms.dba.loadlib</u>	data set name of the load library containing the DMCL and database name table load modules
<u>idms.objlib.user</u>	filename of the user object library
<u>idms.sysidms</u>	filename of the file containing the SYSIDMS parameters
<u>idms.sysctl</u>	filename of the SYSCTL file
<u>nnnn</u>	version number of the DC/UCF system
<u>sysctl</u>	linkname of the SYSCTL file
<u>temp.pch</u>	temporary file containing the ADSOOPTI object module

►► See *CA-IDMS Database Administration*, for more information about SYSIDMS parameters.

Sample BS2000/OSD JCL for local mode: To execute ADSOBSYS in **local mode**, perform the following steps:

1. Remove the ADD-FILE-LINK statement for sysctl
2. Add the following statements:

```

/ADD-FILE-LINK L-NAME=dictdb,F-NAME=idms.appldict.dictdb,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=dloddb,F-NAME=idms.appldict.dloddb,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=dcmmsg,F-NAME=idms.sysmsg.ddldcmmsg,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=sysjrn1,F-NAME=idms.tapejrn1

```

<u>idms.appldict.ddldml</u>	filename of the data dictionary
<u>idms.appldict.ddldclod</u>	filename of the data dictionary load area
<u>idms.sysmsg.ddldcmmsg</u>	filename of the data dictionary message area
<u>idms.tapejrn1</u>	filename of the tape journal file
<u>dictdb</u>	linkname of the data dictionary DDLDML area
<u>dloddb</u>	linkname of the data dictionary load area (DDLDCLOD)
<u>dmsgdb</u>	linkname of the data dictionary message area (DDLDMV/ESAG)
<u>sysjrn1</u>	linkname of the journal file

Note: The program BS2KOBJM is executed to translate the object code generated by the ADSOBSYS program from an IBM format to a SIEMENS format, and to put the resulting object code into the object library specified by the user.

D.4 ADSOBTAT

What it is: ADSOBTAT is a batch utility that allows the application developer to add, modify, and delete entries in the task application table (TAT). For example, ADSOBTAT can be used to update the TAT for a dictionary when an application is migrated to that dictionary.

►► For more information about the TAT, see Chapter 2, “CA-ADS Application Compiler (ADSA).”

Note:

When an application is added, modified, or deleted by using the application compiler, the TAT is automatically updated in the applicable dictionary, and ADSOBTAT is not required. The TAT can also be updated online by using ADSOTATU, as described under 'ADSOTATU' earlier in this appendix.

How it works: At the beginning of an ADSOBTAT run, ADSOBTAT copies the TAT stored either in the load area or, if the load area has no TAT, in the load library. If no TAT exists, ADSOBTAT creates a new TAT if the action is ADD. ADSOBTAT updates the copy of the TAT, based on the control statements provided. At the end of the run, if the control statements contain no errors, ADSOBTAT stores the copy of the TAT in the load area, replacing any previous copy.

ADSOBTAT does not update a TAT's program description element (PDE) to indicate that a new copy of the TAT exists in the load area. If a TAT is updated by ADSOBTAT and then referenced during a single DC/UCF run, the application developer should update the PDE by issuing the following command:

```
DCMT VARY PROGRAM $ACF@TAT NEW COPY
```

►► For more information on the DCMT VARY PROGRAM command, refer to *CA-IDMS System Tasks and Operator Commands*.

ADSOBTAT output: ADSOBTAT produces a listing that displays the card images of all control statements processed. Error messages, if any, are listed under their associated control statements. If an error in any control statement prevents ADSOBTAT from updating the TAT, ADSOBTAT issues the following message:

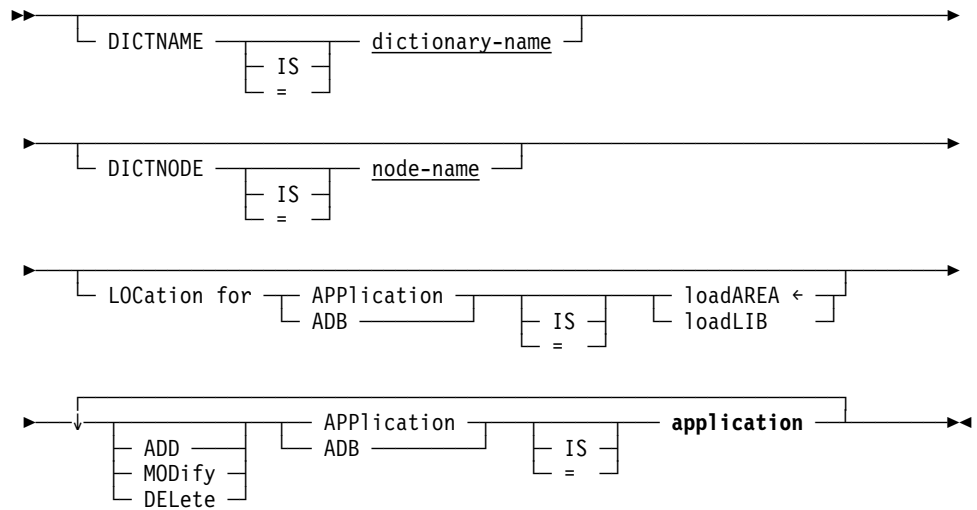
```
DC474029 *** WARNING *** DUE TO ABOVE ERROR TAT WILL NOT BE  
UPDATED DURING THIS RUN; SYNTAX CHECKING ONLY
```

Note: At a site where alternate dictionaries are used, the system database name table must map network subschema IDMSNWKL (used by ADSOBTAT) to the copy of the network subschema appropriate for the alternate dictionary. The database name table is defined by the DBNAME statement.

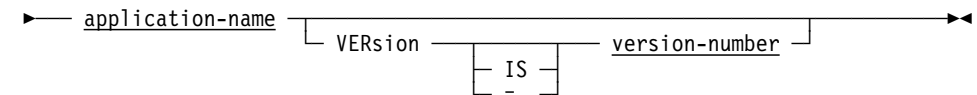
D.4.1 Control statements

Purpose:

Syntax:



Expansion of application



Parameters

DICTNAME IS dictionary-name

Specifies the 1- to 8-character name of the data dictionary in which the TAT is stored.

DICTNAME defaults to the name of the primary dictionary.

DICTNODE IS node-name

Specifies the node that controls the data dictionary in which the TAT is stored.

LOCATION for APPLICATION IS

Introduces where the applications specified in the control statements are stored.

ADB can be used in place of APPLICATION; the equals sign (=) can be used in place of IS.

loadAREA

Specifies that the applications are stored in the load area.

LOADAREA is the default when no location for application is specified.

loadLIB

Specifies that the applications are stored in the load (core-image) library.

The load libraries in which the applications are stored must be specified in the JCL, as follows:

- **OS/390 JCL** -- In the CDMSLIB statement or, if a CDMSLIB statement is not specified, in the STEPLIB statement
- **VSE/ESA JCL** -- In the ASSGN/EXTNT statement for the private core-image library or in the LIBDEF equivalent
- **VM/ESA commands** -- In the GLOBAL LOADLIB command, added to the list of libraries
- **BS2000/OSD JCL** -- In the CDMSLIB chain

ADD

Specifies that task code entries for an application are being added to the TAT.

If ADD is specified and the TAT already contains entries for the application, the action is changed to MOD and a warning message is displayed. If ADD is specified and the TAT does not exist, ADSOBTAT creates a TAT.

ADD is the default if the TAT contains no entries for the application.

MODify

Specifies that the task code entries for an application are being replaced in the TAT by the task codes defined in the current application load module.

If MOD is specified and the TAT does not contain entries for the application, ADSOBTAT treats the request like an ADD request.

MOD is the default if the TAT already contains entries for the application.

DELeTe

Specifies that the task code entries for an application are being deleted from the TAT.

If the TAT does not contain entries for the application, a warning message is issued. Note that the application does not have to exist when DEL is specified.

APPLICATION IS application

Identifies the application.

See expansion of application below.

application-name

Specifies the name of the application.

VERsion is version-number

Gives the version number (in the range 1 through 9999) of the application.

The default version number is 1.

Usage:

Considerations: If specified, the DICTNODE and DICTNAME clauses must be coded first, in any order. The LOCATION clauses, if specified, must be coded next, in any order. The ADD/MODIFY/DELETE APPLICATION clause must be coded last, and can be repeated any number of times to reference several applications.

D.4.2 JCL and commands

JCL for running ADSOBTAT is shown below for OS/390, VSE/ESA, VM/ESA, and BS2000/OSD systems.

D.4.2.1 OS/390 JCL

Sample OS/390 JCL for central version: ADSOBTAT (central version) (OS/390)

```
//ADSOBTAT EXEC PGM=ADSOBTAT,REGION=1024K
//STEPLIB DD DSN=idms.dba.loadlib,DISP=SHR
// DD DSN=idms.loadlib,DISP=SHR
//sysctl DD DSN=idms.sysctl,DISP=SHR
//dcmmsg DD DSN=idms.sysmsg.ddldcmmsg,DISP=SHR
//SYSLST DD SYSOUT=A
//SYSIDMS DD *
DMCL=dmcl-name
Put other SYSIDMS parameters, as appropriate, here
/*
//SYSIPT DD *
Put ADSOBTAT parameters, as appropriate, here
/*
//*
```

ADSOBTAT (local mode) (OS/390)

```
//ADSOBTAT EXEC PGM=ADSOBTAT,REGION=1024K
//STEPLIB DD DSN=idms.dba.loadlib,DISP=SHR
// DD DSN=idms.loadlib,DISP=SHR
//dloddb DD DSN=idms.appldict.ddldclod,DISP=SHR
//dcmmsg DD DSN=idms.sysmsg.ddldcmmsg,DISP=SHR
//sysjrn1 DD DSN=idms.tapejrn1,DISP=(NEW,CATLG),UNIT=tape
//SYSLST DD SYSOUT=A
//SYSIDMS DD *
DMCL=dmcl-name
Put other SYSIDMS parameters, as appropriate, here
/*
//SYSIPT DD *
Put ADSOBTAT parameters, as appropriate, here
/*
//*
```

<u>idms.dba.loadlib</u>	Data set name of the load library containing the DMCL and database name table load modules
<u>idms.loadlib</u>	Data set name of the load library containing the CA-IDMS executable modules
<u>sysctl</u>	DDname of the SYSCTL file
<u>idms.sysctl</u>	Data set name of the SYSCTL file
<u>dcmsg</u>	DDname of the system message (DDLDM/ESAG) area
<u>idms.sysmsg.ddldcmsg</u>	Data set name of the system message (DDLDM/ESAG) area
<u>dmcl-name</u>	Name of the DMCL load module
<u>dloddb</u>	DDname of the application dictionary definition load (DDLDCLOD) area
<u>idms.appldict.ddldclod</u>	Data set name of the application dictionary definition load (DDLDCLOD) area
<u>sysjrn1</u>	DDname of the tape journal file
<u>idms.tapejrn1</u>	Data set name of the tape journal file
<u>tape</u>	symbolic device name of the tape journal file

►► See *CA-IDMS Database Administration*, for more information about SYSIDMS parameters.

D.4.2.2 VSE/ESA JCL

Sample VSE/ESA JCL for central version: ADSOBTAT (VSE/ESA)

```
// UPSI      b                if specified in the IDMSOPTI module
// DLBL      userlib
// EXTENT    ,nnnnnn
// LIBDEF    *,SEARCH=(userlib.cdmslib)
// EXEC      ADSOBTAT
control statements
SYSIDMS parameters
```

<u>b</u>	appropriate 1- through 8-character UPSI bit switch, as specified in the IDMSOPTI module
<u>nnnnnn</u>	volume serial number of the library
<u>userlib</u>	filename of the user library
<u>userlib.cdmslib</u>	file-id of the CA-IDMS sublibrary
<u>SYSIDMS parameters</u>	A list of SYSIDMS parameters for this job

►► See *CA-IDMS Database Administration*, for more information about SYSIDMS parameters.

Sample VSE/ESA JCL for local mode: To execute ADSOBTAT in **local mode**, perform the following steps:

1. Remove the UPSI specification.
2. Add the following statements before the EXEC statement:

```
// DLBL      dloddb,'idms.appldict.ddldclod',,DA
// EXTENT    sys017,nnnnnn
// ASSGN     sys017,DISK,VOL=nnnnnn,SHR
// DLBL      dmsgdb,'idms.sysmsg.ddldcmsg',,DA
// EXTENT    sys016,nnnnnn
// ASSGN     sys016,DISK,VOL=nnnnnn,SHR
// TLBL      sys009,'idms.tapejrn1',,nnnnnn,,f
// ASSGN     sys009,TAPE,VOL=nnnnnn
```

<u>idms.appldict.ddldclod</u>	file-id of the data dictionary load area
<u>idms.sysmsg.ddldcmsg</u>	file-id of the data dictionary message area
<u>idms.tapejrn1</u>	file-id of the tape journal file
<u>dloddb</u>	filename of the data dictionary load area (DDLDCLOD)
<u>dmsgdb</u>	filename of the data dictionary message area (DDLDM/ESAG)
<u>f</u>	file number of the tape journal file
<u>nnnnnn</u>	volume serial number
<u>sys009</u>	logical unit assignment for the tape journal file
<u>sys016</u>	logical unit assignment for the data dictionary message area
<u>sys017</u>	logical unit assignment for data dictionary load area

D.4.2.3 VM/ESA commands

Sample VM/ESA commands for central version: ADSOBTAT (VM/ESA)

```
FILEDEF SYSLST PRINTER
FILEDEF SYSIDMS DISK sysidms input a
FILEDEF SYSIPT DISK btat input a
GLOBAL LOADLIB idmslib
OSRUN ADSOBTAT
```

<u>sysidms input a</u>	filename, filetype, and filemode of the file containing the SYSIDMS input parameters
<u>btat input a</u>	file identifier of the file containing ADSOBTAT source statements
<u>idmslib</u>	filename of the CA-IDMS LOADLIB library

►► See *CA-IDMS Database Administration*, for more information about SYSIDMS parameters.

Sample VM/ESA commands for local mode: To execute ADSOBTAT in **local mode**, add the following commands before the OSRUN command:

```
FILEDEF sysjrn1 TAP1 SL VOLID nnnnnn (RECFM VB LRECL 111 BLKSIZE bbb
FILEDEF dictdb DISK dictdb dictfile d (RECFM F LRECL ppp BLKSIZE ppp XTENT nnn
FILEDEF dloddb DISK dloddb dictfile f (RECFM F LRECL ppp BLKSIZE ppp XTENT nnn
FILEDEF dmsgdb DISK dmsgdb dictfile e (RECFM F LRECL ppp BLKSIZE ppp XTENT nnn
```

<u>bbb</u>	block size of the tape journal file
<u>dictdb</u>	ddname of the data dictionary DDLDDL area
<u>111</u>	record length of the tape journal file
<u>nnnnnn</u>	volume serial number of the tape journal file
<u>sysjrn1</u>	ddname of the tape journal file
<u>dictdb dictfile d</u>	file identifier of the data dictionary DDLDDL area
<u>ppp</u>	page size of the area
<u>dloddb</u>	ddname of the data dictionary load area (DDLDCLOD)
<u>dloddb dictfile f</u>	file identifier of the data dictionary load area
<u>dmsgdb</u>	ddname of the data dictionary message area (DDLDM/ESAG)
<u>dmsgdb dictfile e</u>	file identifier of the data dictionary message area

Specifying central version or local mode: To specify whether ADSOBTAT executes under central version or in local mode, take one of the following actions:

1. Specify either CVMACH=*dc/ucf-machine-name* (for central version) or *LOCAL* (for local mode) as the first statement to submit to ADSOBTAT.
Dc/ucf-machine-name is the 1- through 8-character user identifier of the VM/ESA virtual machine in which the DC/UCF system is executing.
2. Link edit ADSOBTAT with an IDMSOPTI module that specifies either CVMACH=*dc/ucf-machine-name* (for central version) or CENTRAL=NO (for local mode). Instructions for creating an IDMSOPTI module are given in *CA-IDMS System Operations*.
3. Code PARM='CVMACH=*dc/ucf-machine-name*' or PARM='*LOCAL*' on the OSRUN command used to invoke the compiler. This option is not allowed if the OSRUN command is issued from a VM/ESA EXEC program; however, it is allowed if the OSRUN command is issued from a System Product interpreter (REXX) or EXEC 2 program.

Additional information about central version and local mode operations in the VM/ESA environment can be found in *CA-IDMS Installation and Maintenance — VM/ESA*.

D.4.2.4 BS2000/OSD JCL

Sample BS2000/OSD JCL for central version: ADSOBTAT (BS2000/OSD)

```
/ADD-FILE-LINK L-NAME=CDMSLIB,F-NAME=idms.dba.loadlib
/ADD-FILE-LINK L-NAME=CDMSLIB1,F-NAME=idms.loadlib
/ADD-FILE-LINK L-NAME=CDMSLODR,F-NAME=idms.loadlib
/ADD-FILE-LINK L-NAME=sysctl,F-NAME=idms.sysctl,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=SYSIDMS,F-NAME=idms.sysidms
/ASSIGN-SYSDTA TO=*SYSCMD
/START-PROG *MOD(ELEM=ADSOBTAT,LIB=idms.loadlib,RUN-MODE=*ADV)
control statements
```

<u>idms.loadlib</u>	filename of the CA-IDMS load library
<u>idms.dba.loadlib</u>	data set name of the load library containing the DMCL and database name table load modules
<u>idms.sysidms</u>	filename of the file containing the SYSIDMS parameters
<u>idms.sysctl</u>	filename of the SYSCTL file
<u>sysctl</u>	linkname of the SYSCTL file

►► See *CA-IDMS Database Administration*, for more information about SYSIDMS parameters.

Sample BS2000/OSD JCL for local mode: To execute ADSOBTAT in **local mode**, perform the following steps:

1. Remove the ADD-FILE-LINK statement for sysctl
2. Add the following statements:

```
/ADD-FILE-LINK L-NAME=dloddb,F-NAME=idms.appldict.dloddb,SHARED-UPD=*YES  
/ADD-FILE-LINK L-NAME=dcmsg,F-NAME=idms.sysmsg.ddldcmsg,SHARED-UPD=*YES  
/ADD-FILE-LINK L-NAME=sysjrn1,F-NAME=idms.tapejrn1
```

<u>idms.appldict.ddldclod</u>	filename of the data dictionary load area
<u>idms.sysmsg.ddldcmsg</u>	filename of the data dictionary message area
<u>idms.tapejrn1</u>	filename of the tape journal file
<u>dloddb</u>	linkname of the data dictionary load area (DDLDCLOD)
<u>dmsgdb</u>	linkname of the data dictionary message area (DDLDM/ESAG)
<u>sysjrn1</u>	linkname of the tape journal file

D.5 ADSOTATU

ADSOTATU is an online utility that allows the application developer to add, modify, and delete entries in the task application table (TAT). For example, ADSOTATU can be used to update the TAT for a dictionary when an application is migrated to that dictionary.

►► For more information about the TAT, see Chapter 2, “CA-ADS Application Compiler (ADSA).”

Note: When an application is added, modified, or deleted by using the application compiler, the TAT is automatically updated in the applicable dictionary, and ADSOTATU is not required. The TAT can also be updated in batch by using ADSOBTAT, as described under 'ADSOBTAT' later in this appendix.

ADSOTATU is invoked by specifying the task code ADSOTATU at the DC/UCF prompt.

ADSOTATU displays the TAT Update Utility screen on which the application developer specifies the name of the application whose TAT entries are being added, modified, or deleted. Additionally, the application developer can specify the dictionary that contains the TAT, the node that controls the dictionary, the application version number, and the action to take regarding the TAT entries.

Each time the application developer specifies an action, ADSOTATU makes a copy of the TAT stored in either the program pool, load area (if the program pool has no TAT), or load library (if the load area has no TAT). If no TAT exists, ADSOTATU creates a new TAT if the action is ADD. ADSOTATU updates the copy of the TAT as appropriate; stores the copy in the load area, replacing any previous TAT; and issues a DCMT VARY PROGRAM NEW COPY command to update the TAT in the program pool.

Specifying activities: The application developer specifies activities in an ADSOTATU session by using the ENTER, CLEAR, and PF9 keys, as follows:

- **ENTER** instructs ADSOTATU to add, modify, or delete entries in the TAT, based on information specified on the screen.

If the TAT is updated successfully, ADSOTATU issues a confirming message. The screen can be used repeatedly to specify several applications.

If ADSOTATU encounters an error, it redisplay the screen with an appropriate error message. The application developer can change information on the screen, then resubmit the information by pressing ENTER.

- **CLEAR** and **PF9** terminate ADSOTATU and return control to DC/UCF.

D.5.1 TAT update utility screen

Sample screen

CA-ADS REL 15.0		COMPUTER ASSOCIATES INTERNATIONAL, INC.		***TAT UPDATE UTILITY***	
ACTION: (ADD/MOD/DEL)		DICT NAME:		NODE:	
APPLICATION:		VERSION:			

Field descriptions:

DICT NAME: Specifies the 1- to 8-character name of the data dictionary in which the TAT is stored.

DICT NAME defaults to the name of the primary dictionary. Specifying a dictionary name is equivalent to issuing a DCUF SET DICTNAME command under DC/UCF.

NODE: (for DDS only) Specifies the DDS node that controls the data dictionary specified by DICT NAME.

NODE defaults to the system currently in use. Specifying a node name is equivalent to issuing a DCUF SET DICTNODE command under DC/UCF.

ADD: Specifies that task code entries for an application are being added to the TAT.

If ADD is specified and the TAT already contains entries for the application, the action is changed to MOD and a warning message is displayed.

ADD is the default if the TAT contains no entries for the application.

MODify: Specifies that the task code entries for an application are being replaced in the TAT by the task codes defined in the current application load module.

If MODIFY is specified and the TAT does not contain entries for the application, ADSOTATU treats the request like an ADD request.

MODIFY is the default if the TAT already contains entries for the application.

DELeTe: Specifies that the task code entries for an application are being deleted from the TAT.

If the TAT does not contain entries for the application, a warning message is issued. Note that the application does not have to exist when DEL is specified.

APPLICATION: Specifies the name of the application. If the action is ADD or MOD, the specified application must exist in the data dictionary specified by DICT NAME.

VERSION: Specifies the version number (in the range 1 through 9999) of the application. The default version number is 1.

Appendix E. Activity Logging for a CA-ADS Dialog

E.1 Overview	E-3
E.2 Data dictionary organization	E-4
E.3 Activity logging record formats	E-5

E.1 Overview

What it does: The activity logging feature of CA-ADS creates activity records that document all potential database activity for a dialog. Documentation is based on the database commands issued explicitly or implicitly by the dialog's processes. (Examples of implicit database commands are the implicit READY command, issued automatically for each subschema area when a run unit is opened for a process, and the BIND command, issued automatically for each subschema record used by the dialog.) If enabled, activity logging is performed when a dialog is compiled and has no impact on runtime performance.

The activity logging feature can be used to perform the following tasks:

- **Monitor database usage** If runtime activity is high for a particular subschema area, set, record, or logical record, activity records can show which dialogs contain database commands that potentially access the entity.
- **Modify dictionary entity occurrences** If a subschema area, set, record, or logical record needs to be modified, activity records can show which dialogs need to be recompiled as a result of the modification.

Enabling activity logging: The activity logging feature is enabled or disabled at system generation. The application developer can override the system generation default when defining a dialog.

►► For more information, see Chapter 3, “CA-ADS Dialog Compiler (ADSC).”

If enabled, the activity logging feature creates database activity records when a dialog is compiled. Database activity records can be accessed by using query programs such as OnLine Query, by using the Data Dictionary Reporter, or by writing an appropriate program.

The remainder of this appendix describes the data dictionary organization and the format of activity logging records.

E.2 Data dictionary organization

Database activity records are stored as junction records between a dialog's PROG-051 record and the dictionary's AREA, SET, RECORD, and LOGICAL RECORD entities, as follows:

Dictionary entity	Junction record stored
SSA-024 (AREA)	AFACT-057
SSOR-034 (SET)	SETACT-061
SSR-032 (RECORD)	RCDACT-059
LR-190 (LOGICAL RECORD)	LRACT-193

Thus, if an area, set, record, or logical record must be modified, the application developer can follow a path from the dictionary entity occurrence, through the appropriate junction record, to the PROG-051 records of the dialogs that need to be recompiled because of the modification.

E.3 Activity logging record formats

An activity logging record contains the following information about the database command being logged for a dialog:

- The function number of the database command being logged
- The number of times in the dialog that the database command is coded against the dictionary entity occurrence
- The name of the dictionary entity occurrence

AFACT-057: The record description of the AFACT-057 junction record is as follows:

02 AF-FUNCT-057	PICTURE IS S9(4) USAGE IS COMP.
02 AF-COUNT-057	PICTURE IS 9(4) USAGE IS COMP.
02 AF-AREA-OWN-057	PICTURE IS X(32) USAGE IS DISPLAY.
02 EXTRNL-NAME-057	PICTURE IS X(32) USAGE IS DISPLAY.
02 FILLER	PICTURE IS X(4) USAGE IS DISPLAY.

SETACT-061: The format of the SETACT-061 junction record is as follows:

02 SA-FUNCT-061	PICTURE IS 9(4) USAGE IS COMP.
02 SA-COUNT-061	PICTURE IS 9(4) USAGE IS COMP.
02 SA-SET-OWN-061	PICTURE IS X(32) USAGE IS DISPLAY.
02 FILLER	PICTURE IS X(4) USAGE IS DISPLAY.

RCDACT-059: The format of the RCDACT-059 junction record is as follows:

02 RA-FUNCT-059	PICTURE IS 9(4) USAGE IS COMP.
02 RA-COUNT--059	PICTURE IS 9(4) USAGE IS COMP.
02 RA-RCD-OWN-059	PICTURE IS X(32) USAGE IS DISPLAY.
02 FILLER	PICTURE IS X(4) USAGE IS DISPLAY.

LRACT-193: The format of the LRACT-193 junction record is as follows:

02 FUNCT-193	PICTURE IS S9(4) USAGE IS COMP.
02 COUNT-193	PICTURE IS S9(4) USAGE IS COMP.
02 LR-NAM-193	PICTURE IS X(16) USAGE IS DISPLAY.

Record fields

FUNCT

Contains the numeric function number that is assigned to the database command or logical record command being logged.

The function numbers for the AFACT-057 (AREA), SETACT-061 (SET), RCDACT-059 (RECORD), and LRACT-193 (LOGICAL RECORD) junction records and their associated database or logical record commands are listed in the following table.

Note: No activity records are stored for the COMMIT and ROLLBACK database commands.

COUNT

Contains the number of times the logged database command is coded in all of the processes of the dialog.

OWN

Contains the name of the record, set, or area whose activity is being documented by the record, set, or area activity record.

EXTRNL-NAME

Contains spaces.

NAM

Contains the name of the logical record whose activity is being documented by the logical record activity record.

Usage:

Considerations: The COUNT field for a READY command reflects only the effective READY commands issued implicitly or explicitly in the dialog's processes. An implicit or explicit READY command sets the usage mode of a database area during a dialog's premap or response process.

►► For more information, see Chapter 17, “Map Commands.”

If the same area is named in more than one READY command in a process, the usage mode specified in the last READY command applies to the named area for the entire process. The COUNT field of a junction record for a READY command reflects the number of processes for which the specified usage mode applies to the specified area.

Activity logging function numbers and associated commands

Junction record	Function number	Navigational or LRF database command
AFACT-057	3	FIND
	6	KEEP and KEEP LONGTERM
	15	ACCEPT
	23	FIND KEEP
	36	READY USAGE MODE UPDATE
	37	READY USAGE MODE RETRIEVAL
	38	READY USAGE MODE PROTECTED UPDATE
	39	READY USAGE MODE PROTECTED RETRIEVAL

Junction record	Function number	Navigational or LRF database command
	40	READY USAGE MODE EXCLUSIVE RETRIEVAL
	41	READY USAGE MODE EXCLUSIVE UPDATE
	43	OBTAIN
	63	OBTAIN KEEP
SETACT-061	3	FIND
	6	KEEP and KEEP LONGTERM
	7	CONNECT
	11	DISCONNECT
	15	ACCEPT
	16	IF SET EMPTY/MEMBER
	17	RETURN
	23	FIND KEEP
	43	OBTAIN
	63	OBTAIN KEEP
RCDACT-059	2	ERASE
	3	FIND
	5	GET
	6	KEEP and KEEP LONGTERM
	7	CONNECT
	8	MODIFY
	11	DISCONNECT
	12	STORE
	14	BIND
	15	ACCEPT
	23	FIND KEEP
	43	OBTAIN
	63	OBTAIN KEEP
LRACT-193	2	ERASE
	8	MODIFY

Junction record	Function number	Navigational or LRF database command
	12	STORE
	43	OBTAIN

Appendix F. Built-in Function Support

- F.1 Overview F-3
- F.2 Internal structure of built-in functions F-4
 - F.2.1 Master function table F-5
 - F.2.2 Model XDE module F-6
 - F.2.3 XDEs and VXDEs F-8
 - F.2.4 Processing program modules F-17
 - F.2.5 Runtime processing of built-in functions F-24
- F.3 Assembler macros F-27
 - F.3.1 #EFUNMST F-27
 - F.3.2 RHDCEVBF F-28
 - F.3.3 #EFUNMOD F-31
- F.4 Changing invocation names F-40
- F.5 Creating user-defined built-in functions F-41
 - F.5.1 Steps for generating a user-defined built-in function F-41
 - F.5.2 LRF considerations for user-defined built-in functions F-42
 - F.5.3 Calling a user-defined built-in function F-42

F.1 Overview

About built-in functions: CA-ADS built-in function support enables an installation to change the invocation names of built-in functions and to generate user-defined built-in functions. This appendix discusses the following topics:

- The internal structure of built-in functions
- The assembler macros that define components of built-in functions
- How to change invocation names
- How to create user-defined built-in functions

To change invocation names, the user needs only to read the following topics:

- The discussion of the master function table under 'Internal Structure of Built-in Functions'
- The discussion of the #EFUNMST macro under 'Assembler Macros'
- The instructions provided under 'Changing Invocation Names'

F.2 Internal structure of built-in functions

CA-ADS supplied built-in functions and user-defined built-in functions share the same internal structure. This structure consists of the following components.

Master function table: The **master function table** lists the invocation names for each built-in function. The master function table is used during process compilation to associate a coded invocation name with a real (generic) function name and to point to a model XDE (expression description element) module that describes the function.

Model XDE modules: **Model XDE modules** contain one or more model XDE tables. Each model XDE table describes a function, including the function's parameters, work area requirements, result field, and processing program name. During process compilation, a model XDE table is used to produce a series of XDEs that form the compiled representation of the function.

XDEs and VSDEs: **XDEs** and **VXDEs** describe functions at runtime. XDEs are created during process compilation; one VXDE (variable expression description element) is created for each XDE at runtime to hold variable information.

Processing program modules: **Processing program modules** contain processing logic for one or more functions. At runtime, when the XDEs and VXDEs for the function are processed, the runtime system calls the appropriate program and passes to it all required information. The program executes, then returns control to the dialog.

F.2.1 Master function table

The master function table is a dictionary load module that lists the invocation names for all CA-ADS supplied and user defined built-in functions. Each entry contains a function invocation name, a corresponding real (generic) function name, and the name of the model XDE module that describes the function.

The concatenate function, for example, has by default three invocation names: CONCATENATE, CONCAT, and CON. Each invocation name has an entry in the master function table. Each entry also specifies the real function name for the concatenate function, CONCAT, and the model XDE module that describes the concatenate function, RHDCEV51.

During compilation of a coded function, the dialog compiler searches the master function table for the coded invocation name. If it finds an entry, it uses the information in the entry to find the model XDE module that describes the function; if it does not find an entry, it generates a syntax error message.

Note: At runtime, an invocation name that is used in a dialog must not duplicate the name of a record element known to the dialog. If it does, CA-ADS interprets the function as a subscripted reference to the record element.

The DSECT for an entry in the master function table is shown below. The load module for the master function table is stored in the data dictionary load area under the name RHDCEVBF.

DSECT for a master function table entry

EFMASDS	DSECT	11:15:30 03/06/86	00001000
*		EVAL MASTER FUNCTION TABLE ENTRY DSECT	00002000
EFMINAML DS	H	LENGTH OF INVOCATION FUNCT NAME	00003000
EFMINAME DS	CL32	FUNCTION NAME - INVOCATION	00004000
EFMRNAME DS	CL8	FUNCTION NAME - REAL	00005000
EFMMPGMN DS	CL8	PROGRAM NAME - MODEL XDE TABLE	00006000
EFMMPGMV DS	H	PROGRAM VERSION - MODEL XDE TABLE	00007000
EFMFLAG1 DS	XL1	MASTER FUNCTION ENTRY FLAG1	00008000
EFMASFU EQU	X'80'	AGGREGATE FUNCTION ENTRY	00009000
	DS XL3	FILLER	00010000
EFMASLNG EQU	*-EFMASDS	LENGTH OF MASTER ENTRY	00011000

F.2.2 Model XDE module

A model XDE module is a load module that contains one or more model XDE tables, each describing a function. During process compilation of a function, the dialog compiler uses the appropriate model XDE table to generate a series of XDEs that form the compiled representation of the function. A model XDE table contains the following entries:

- A **header** entry that contains the function's processing program name, work area requirements, number and types of function parameters, and a description of the function's result field. Each model XDE table contains one header entry.
- **XDE** entries that describe the function's parameters and determine certain characteristics of the result field. One XDE entry exists for each function parameter.
- **Data type conversion** entries that define the data types and length of each function parameter. One or more data type conversion entries exist for each function parameter.

The DSECTs for these three entries are shown below. The load modules for the model XDE modules are stored in the load library. The model XDE source and load modules for the CA-ADS supplied built-in functions are called RHDCEV51, RHDCEV52, RHDCEV53, and RHDCEV59, and can be used as a reference when defining user-defined built-in functions.

DSECTs for the model XDE table entries

EFHDRDS	DSECT	12:01:34 05/18/84	00001000
*		EVAL FUNCTION MODEL TABLE HEADER DSECT	00002000
EFHNEXT	DS H	OFFSET TO NEXT HDR ENTRY	00003000
EFHFUNNM	DS CL8	FUNCTION NAME - REAL	00004000
EFHPPGMN	DS CL8	PROCESSING PROGRAM NAME	00005000
EFHPPGMV	DS H	PROCESSING PROGRAM VERSION	00006000
EFHFUNCN	DS XL1	FUNCTION NUMBER	00007000
	DS XL1	FILLER	00008000
EFHWORKL	DS H	LENGTH OF REQUIRED WORKAREA	00009000
EFHZOPND	DS 0XL4	4 X'00'S INDICATE ZERO OPERANDS	00010000
EFHFOPDN	DS H	NUMBER OF FIXED OPERANDS	00011000
EFHVOPDO	DS H	OFFSET TO VARIABLE OPERAND MODEL	00012000
EFHRESLN	DS H	RESULT LENGTH IN BYTES	00013000
EFHRDATP	DS XL1	RESULT DATA TYPE	00014000
EFHRNDEC	DS XL1	RESULT NUMBER DECIMALS	00015000
	DS XL4	FILLER	00016000
EFHDLRNG	EQU *-EFHDRDS	LENGTH OF FUNCTION MODEL HEADER	00017000
EFXDEDS	DSECT	07:36:43 05/31/84	00001000
*		EVAL FUNCTION MODEL XDE DSECT	00002000
EFXNEXT	DS H	OFFSET TO NEXT MODEL XDE	00003000
EFXNDEC	DS XL1	NUMBER OF DECIMALS	00004000
EFXRLCF	DS XL1	RESULT LENGTH CALCULATION FLAG	00005000
EFXRSCP	EQU X'80'	ADD LENGTH	00006000
EFXRCLS	EQU X'40'	SUBT LENGTH	00007000
*		IF ZERO, IGNORE	00008000
EFXFLAG1	DS XL1	FIRST FLAG	00009000
EFXF1MAN	EQU X'80'	ON=MANDATORY, OFF=OPTIONAL	00010000
EFXF1TRU	EQU X'40'	ON=TRUNCATE, OFF=ROUND	00011000
EFXF1RES	EQU X'20'	RESULT CHARACTERISTICS DEFAULT	00012000
	SPACE 1		00013000
	DS XL3	FILLER	00014000
EFXDCTN	DS H	NUMBER OF ENTRIES IN DATA CONV TBL	00014500
EFXLNG1	EQU *-EFXDEDS	BASE LENGTH OF ENTRY	00015000
	SPACE 1		00016000
*		DATA TYPE CONVERSION TABLE	00017000
EFXCNVE	DSECT	CONVERSION TBL ENTRY DSECT	00018000
EFXSRCT	DS XL1	SOURCE DATATYPE	00019000
EFXTART	DS XL1	TARGET DATATYPE	00020000
EFXTARL	DS H	TARGET LENGTH	00021000
	DS XL2	FILLER	00022000
	SPACE 1		00023000
EFXDCTL	EQU *-EFXSRCT	LENGTH OF ENTRY	00024000

F.2.3 XDEs and VXDEs

XDEs (expression description elements) and VXDEs (variable expression description elements) form the compiled representation of a process at runtime. During compilation, each process statement is converted into a series of XDEs that represent the operands and operations within each statement. The XDEs of the statements are strung together to form the compiled representation of the process. At runtime, the runtime system builds a VXDE for each XDE. VXDEs contain variable runtime information; the information in XDEs does not change.

The compiled representation of a function consists of one operand XDE/VXDE for each parameter and one function XDE/VXDE for the function. These XDE/VXDE pairs contain the following information:

- **Function XDE/VXDE:**
 - Name and address of the function's processing program module
 - Function number identifying the appropriate program within the processing program module
 - Number of function operands (parameters)
 - Address of the work area available to the processing program
 - Description and address of the function's result field
 - Address of the operand VXDE for the last parameter in the parameter list
- **Operand XDE/VXDE:**
 - Description and address of the operand (parameter)
 - Address of the operand VXDE for the previous parameter in the parameter list

The DSECTs for the XDE and VXDE are shown below.

►► The runtime use of the XDEs and VXDEs is described under "Runtime Processing" later in this appendix.

DSECT of the XDE (Expression Description Element)

```

SPACE 1
* * * * *
*      #AXDEDS - EXPRESSION DESCRIPTION ELEMENT      *
* * * * *
* THIS COPY MEMBER IS INCLUDED IN #XDEDS. IF ANY CHANGES ARE *
* MADE HERE, PLEASE INSURE THAT ALL MODULES CONTAINING #XDEDS ARE *
* REASSEMBLED. *
* *
* THIS MACRO ALLOWS THE ADS MODULES TO MORE EASILY REFERENCE XDE *
* FIELDS WHEN AN XDE IS BEING BUILT WITHOUT THE XDENEXT FIELD. *
* * * * *
SPACE
XDEDATAD DS    0F      (REAL) DATA ADDRESS (OPERAND OR RESULT)
XDEBRNXT DS    0F      ** BRANCH OPERATOR XDES ONLY
*
* IF RELOCATABLE XDE MODE TO BE USED IN
* EVAL/ADSOXDES THE OFFSET
* OF BRANCH TARGET XDE
* FROM 1ST XDE ELSE REAL ADDRESS OF TARGET
XDEDTABO DS    H      (LOGICAL) DATA ADDRESS - ADCON TABLE OFFSET
XDEDDSPL DS    H      (LOGICAL) DATA ADDRESS - DISPLACEMENT
* NOTE THAT REAL ADDRESSES ARE DISTINGUISHED FROM LOGICAL ADDRESSES
* (TABLE OFFSET/DISPLACEMENT PAIRS) BY THE X'80' BIT OF THE HIGH
* ORDER ADDRESS BYTE - ON => REAL, OFF => LOGICAL.
SPACE
XDETGTT EQU    X'80'   High order bit of DYN used as temporary
*                      flag during executable code generation
*                      with following meaning :
*                      ON  -> This XDE is a target of a BRC2 XDE
*                      OFF -> This XDE is a NOT a BRC2 target
* This bit will ALWAYS be OFF in ALL XDEs
* in a final FDB.
XDEDYN  DS    H      OFFSET INTO DYNAMIC AREA OF VXDE
XDEDATLN DS    0H     OPERAND LENGTH (IN BYTES)
XDEBROFF DS    0H     ** BRANCH OPERATOR XDES ONLY
*                      OFFSET IN XDES OF BRANCH TARGET FROM
*                      BRANCH OPERATOR. ONLY USED WHEN CONTIG.
*                      XDE MODE USED IN EVAL.
*                      ALWAYS USED IN ADSOXDES.
XDEBITDP DS    C      BIT DISPLACEMENT (FOR MB-BIN)
XDEBITLN DS    C      LEN (NBR OF BITS)(FOR MB-BIN)
*
*                      ORG XDEDATLN
XDEEPPWR DS    C      POWER 10-1 (FOR EDIT)
XDEEPLN  DS    0C     PICTURE LENGTH (FOR EDIT)
XDEESGN  DS    C      SIGN CHARACTER (FOR EDIT)
XDENODEC DS    C      NUMBER OF DECIMAL PLACES
XDEDATYP DS    C      OPERAND DATA TYPE
SPACE

```

```

*   XDE DATA TYPE EQUATES :
XDEGRP EQU 0      GROUP
XDEEBCD EQU 1      EBCDIC
XDEHBBIN EQU 2      BINARY HALFWORD
XDEFBIN EQU 3      BINARY FULLWORD
XDESPAK EQU 4      PACKED DECIMAL (SIGNED)
XDEDUPAK EQU 5      PACKED DECIMAL (UNSIGNED)
XDEDSZON EQU 6      ZONED DECIMAL (SIGNED)
XDEDUZON EQU 7      ZONED DECIMAL (UNSIGNED)
XDEDFLTD EQU 8      DISPLAY FLOATING POINT
XDEDSFLT EQU 9      INTERNAL FLOAT (SHORT)
XDEDLFLT EQU 10     INTERNAL FLOAT (LONG)
XDEDBIT EQU 11     BIT
XDEDBBIN EQU 12     BINARY DOUBLEWORD
XDEDFC EQU 13      FIGURATIVE CONSTANT
XDEDMBIN EQU 14     MULTI-BIT BINARY (PL1 STYLE)
XDEVVCHR EQU 15     VARYING CHARACTER
XDEDEDIT EQU 16     EDIT INFO
XDEDEDP EQU 17     EDIT PICTURE
XDEGEXT EQU 18     EXTERNAL GRAPHICS SO.....SI
XDEGINT EQU 19     INTERNAL GRAPHICS
* FOLLOWING EQU SHOULD ALWAYS REFLECT THE HIGHEST DATA TYPE
* !!!!!!!!!!!!!!! PLEASE NOTE : !!!!!!!!!!!!!!!
* !!!! ANY CHANGE TO THE FOLLOWING EQUATE REQUIRES CHGS !!!
* !!!! TO RHDCEVAL AND ADSOXDES.
* !!!!!!!!!!!!!!!
XDEDMXTP EQU XDEGINT MAX DATA TYPE VALUE
* NOTE THAT RHDCEVAL CURRENTLY ONLY SUPPORTS BIT FIELDS IN LOGICAL
* OPERATIONS.
    SPACE
XDEOPTYP DS 0C      OPERATION/OPERAND CODE
XDEEPAD DS C        PAD CHARACTER (FOR EDIT)
    SPACE
*   XDE OPERATOR TYPE EQUATES :
XDEOOPND EQU 0      OPERAND (NOT OPERATOR)
XDEOPNOT EQU 5      LOGICAL "NOT"
XDEOPOR EQU 6      LOGICAL "OR"
XDEOPAND EQU 7      LOGICAL "AND"
XDEOPCNJ EQU 9      CLASS OF LOGICAL CONJUNCTIONS
XDEOPEQ EQU 10     "EQ" (RELATIONAL OPERATION)
XDEOPNE EQU 11     "NE"
XDEOPLT EQU 12     "LT"
XDEOPLE EQU 13     "LE"
XDEOPGT EQU 14     "GT"
XDEOPGE EQU 15     "GE"
XDEOMTCH EQU 16     "MATCHES"
XDEOPCON EQU 17     "CONTAINS"
XDEOPCMP EQU 18     "COMPARE"; 8:EQUAL, 4:<, 2:>

```

```

XDEOPREL EQU 19          CLASS OF RELATIONAL OPERATORS
XDEOUNMN EQU 20          UNARY "-"
XDEOADDN EQU 21          "+"
XDEOBNMN EQU 22          BINARY "-"
XDEOMULT EQU 23          "*"
XDEOPDIV EQU 24          "/"
XDEOPDVR EQU 25          "/" WITH REMAINDER
XDEOPART EQU 29          CLASS OF ARITHMETIC OPERATORS
*   NOTE : Test Under Mask used as pseudo-operator to
*           process "boolean variables", e.g. Map Status tests.
*           This operator will never appear directly
*           in an XDE list.
XDEOPTM EQU 35           "TEST UNDER MASK"
XDEOASGN EQU 40          "ASSIGNMENT"
*   NOTE : MULTIPLE ASSIGNMENT NO LONGER SUPPORTED
*           Operator code available for reuse.
*DEOASGM EQU 41          "ASSIGNMENT",MULTIPLE TARGETS
XDEOASGR EQU 42          REVERSE ASSIGNMENT
XDEOINDX EQU 45          ARRAY "INDEX"
*
*   NOTE : DATE CONVERSIONS NO LONGER SUPPORTED.
*           THESE CODES ARE AVAILABLE FOR REUSE.
*
*DEODTJA EQU 50          JULIAN DATE TO GREGORIAN (AMER.
*DEODTJG EQU 51          JULIAN DATE TO GREGORIAN (WORLD
*DEODTAJ EQU 52          GREGORIAN DATE (AMER. - MMDDYY)
*DEODTGJ EQU 53          GREGORIAN DATE (WORLD - DDMYY)
*DEODTES EQU 55          CLASS OF DATE CONVERSIONS
*   NOTE : CONCATENATE NO LONGER SUPPORTED.
*           Operator code available for reuse.

```

```
*DEOCONC EQU 60          "CONCATENATION"
*  NOTE : INDA ONLY USED BY DEBUGGER.
*        Not supported by ADSOXDES.
XDEOINDA EQU 65          INDIRECT ADDRESSING
** Following is a "pseudo-opcode" used only in generation
** of machine code for True/False DXBs. This opcode will NEVER
** appear in an XDE list. It is used only for convenience
** so that generating code for T/F DXBs easily fits into the
** standard methodology of code generation.
XDEODXTF EQU 78          True/False DXB
XDEOBRC2 EQU 79          BRANCH OPERATOR
XDEOBRCH EQU 80          BRANCH OPERATOR
*  NOTE : IF A NEW OPERATOR CODE IS ADDED WHICH IS
*          GREATER THAN THE CURRENT VALUE OF XDEOMXTP,
*          WE MUST CHANGE XDEOMXTP TO THIS NEW VALUE.
*          IN THIS CASE, WE MUST ALSO ADD ENTRIES TO THE
*          RHDCEVAL/ADSOXDES OPTABLE.
*          TO MINIMIZE THE SIZE OF OPTABLE,
*          IT WOULD BE BEST TO ASSIGN NEW CODES <=
*          THE CURRENT VALUE OF XDEOMXTP.
*          RHDCEVAL/ADSOXDES CURRENTLY ASSUME THAT THE ONLY
*          VALID OPTYP > XDEOMXTP IS XDEOUFUN AND THIS IS
*          HANDLED AS A SPECIAL CASE.
```

XDEOMXTP EQU	XDEOBRCH	MAX OPTYP VALID FOR USE WITH
*		RHDCEVAL/ADSOXDES OPTABLE.
XDEADSLR EQU	253	"OF LR" OPERAND (USED BY ADS/ONLINE)
*		CODE NOT USED FOR AN OPERATOR
XDEOKWD EQU	254	KEYWORD - USED BY LRF.
*		ALL OTHERS TREAT AS OPERAND
XDEOUFUN EQU	255	USER-DEFINED FUNCTION
XDEFLAG DS	C	FLAG BYTE
XDEFNVL EQU	X'80'	FIELD IS NOT VALUED
XDEFNED EQU	X'40'	NO DATA VALIDATION NEEDED
*		(FOR PACKED/ZONED FIELDS)
XDEFNCV EQU	X'20'	NO CONVERSION NEEDED
XDEADDR EQU	X'10'	XDEDATAD IS OPRND,VS OPRND ADR
*		XDEADDR USED ONLY BY DEBUGGER
XDEFFCZ EQU	X'08'	FIGURATIVE CONSTANT ZERO
XDEBTF EQU	X'08'	** BRANCH OPERATOR XDES ONLY
*		ON => BRANCH IF PREVIOUS RESULT TRUE
*		OFF => BRANCH IF PREVIOUS RESULT FALSE
XDEBNEG EQU	X'04'	** BRANCH2 OPERATOR XDES ONLY
*		If we branch to last XDE, final
*		result is value at top of XDE stack.
*		ON => This value must be negated
*		OFF => Value is correct as is
XDEBEND EQU	X'02'	** BRANCH2 OPERATOR XDES ONLY
*		Special flag for branch on last XDE
*		in list to say final value must be
*		negated.
XDEFTRUN EQU	X'02'	TRUNCATE IF DST DEC < SRC DEC
XDEFRQST EQU	X'01'	USED BY LOGICAL RECORD PROCESSING
	SPACE 1	
*		FLAG CODES FOR EDIT
XDEEF99 EQU	X'01'	SIGNIFICANCE ON HI ORDER
XDEEFLT EQU	X'04'	FLOAT THE SIGN CHARACTER
*		Blank on zero flag never utilized by any EVAL callers
*DEEFBZ EQU	X'08'	BLANK ON ZERO
XDEEFPI EQU	X'10'	XDEEADR POINTS TO PICTURE
XDEEFJL EQU	X'20'	LEFT JUSTIFY OUTPUT
XDEEFNE EQU	X'40'	PICTURE IS ALL X'S
XDEEFNS EQU	X'80'	DO NOT SCALE SOURCE
*		

```

XDELEN1 EQU *-XDE          LENGTH OF STANDARD XDE
        SPACE
*   FOR USER DEFINED FUNCTIONS, SEVERAL ADDITION FLDS ARE REQUIRED
XDEUPGMN DS    CL8          PROGRAM NAME
XDEUNOPS DS    XL1          NBR OPERANDS
XDEUFUNC DS    XL1          FUNCTION NUMBER
XDEUSTLN DS    H            REQ'D STORAGE LENGTH
XDEUPGMV DS    H            PROGRAM VERSION
XDEUFLG1 DS    XL1          USER FUNCTION FLAG BYTE
XDEUAGFU EQU   X'80'        AGGREGATE FUNCTION
        DS    XL1          UNUSED
XDELEN2 EQU   *-XDE          LENGTH OF "USER FUNCTION" XDE
        SPACE
*   EQUATES FOR RESULTS OF COMPARISONS
XDECMPEQ EQU   X'08'        RESULT OF COMPARE IS =
XDECMPLT EQU   X'04'        RESULT OF COMPARE IS <
XDECMPGT EQU   X'02'        RESULT OF COMPARE IS >
        SPACE

```

DSECT of the VXDE (Variable Expression Description Element)

```

* * * * *
*   THE VXDE IS THE DYNAMIC (WRITABLE) PORTION OF THE XDE   *
* * * * *
VXDE      DSECT                      11:16:59 04/14/87
          SPACE
VXDEFLAG DS      0C                FLAG BIT FOR NON-VALUED RESULT
VXDEFNVL EQU     X'80'            NON-VALUED RESULT
VXDESNXT DS      F                OPERAND STACK NEXT XDE ADDR
VXDEFLG2 DS      0C                FLAG BIT FOR ALREADY VALIDATED DECIMAL
VXDEFNED EQU     X'80'            ALREADY VALIDATED DECIMAL
VXDEXDEA DS      F                CORRESPONDING XDE ADDRESS
VXDEDADR DS      F                REAL DATA FIELD ADDRESS
VXDEDLEN EQU     *-VXDE
          SPACE 1

```

```

*      FOR USER-DEFINED FUNCTIONS, THE FOLLOWING FLDS
*      ARE ALSO REQUIRED.
      SPACE 1
VXDEUPGA DS    F          PROGRAM ADDR
VXDEUWKA DS    0F         WORK AREA ADDR
VXDEUWTO DS    H          LOGICAL ADDR TBL OFFSET
VXDEUWDS DS    H          LOGICAL ADDR DISPL
VXDEUFLG DS    CL1        FLAG FOR USER FUNCTIONS
VXDEUBRK EQU   X'80'      AGGREGATE FUNCTION BREAK
VXDEUINT EQU   X'40'      AGGREGATE FUNCTION INIT
VXDEUNIV EQU   X'20'      NO INITIAL VALUE FOR AGG FUN BREAK
VXDEUBAD EQU   X'10'      BAD DATA WITHIN BREAK
VXDEUOVR EQU   X'08'      OVERFLOW WITHIN BREAK
VXDELOD #FLAG X'04'      USER PGM WAS #LOADED
      DS    XL3          UNUSED
VXDEDLN2 EQU   *-VXDE     LNG OF EXTENSION
      SPACE
* * * * *
*   THE XDEIX IS THE DOPE VECTOR USED IN "INDEX" OPERATIONS TO DEFINE *
*   THE FORMAT OF THE ARRAY DIMENSIONS BEING ADDRESSED               *
* * * * *
XDEIX    DSECT
XDEIXDOA DS    F          "DEPEND ON" CONTROL FIELD ADDR
XDEIXNDM DS    H          NUMBER OF DIMENSIONS (IN ARRAY)
XDEIXFLG DS    C          FLAG BYTE
XDEIXFDF EQU   X'80'      FULLWORD "DEPENDS ON" CONTROL FIELD
XDEIXFDH EQU   X'40'      HALFWORD "DEPENDS ON" CONTROL FIELD
      DS    C          UNUSED BYTE
XDEIXRLN EQU   *-XDEIX    LENGTH OF DDOPE VECTOR ROOT
      SPACE
*   THE FOLLOWING FIELDS ARE REPEATED ONCE FOR EACH DIMENSION IN
*   THE ARRAY - FOR MBB TABLES, OFFSET AND SIZE ARE IN BITS
      SPACE
XDEIXOFF DS    H          FIELD OFFSET WITHIN CONTAINING OCCURRENCE
XDEIXSIZ DS    H          SIZE OF A DIMENSION OCCURRENCE
XDEIXMAX DS    H          MAXIMUM SUBSCRIPT VALUE FOR DIMENSION
XDEIXDLN EQU   *-XDEIXOFF LENGTH OF ONE DIMENSION DESCRIPTOR
XDEIXLMT EQU   15         Maximum number of dimensions supported by ADS
      EJECT

```


F.2.4 Processing program modules

Processing program modules contain one or more programs; each program processes one function. When a function XDE/VXDE is processed at runtime, the runtime system calls the appropriate processing program module. The module performs the operation, then returns control to the runtime system. The components of the source module RHDCEV01, which contains the processing programs for the CA-ADS supplied string functions are shown below.

Processing program load modules are usually stored in the load library. The load modules for the CA-ADS supplied built-in functions are named RHDCEV01, RHDCEV02, RHDCEV03, and RHDCEV09.

Components of processing program module RHDCEV01

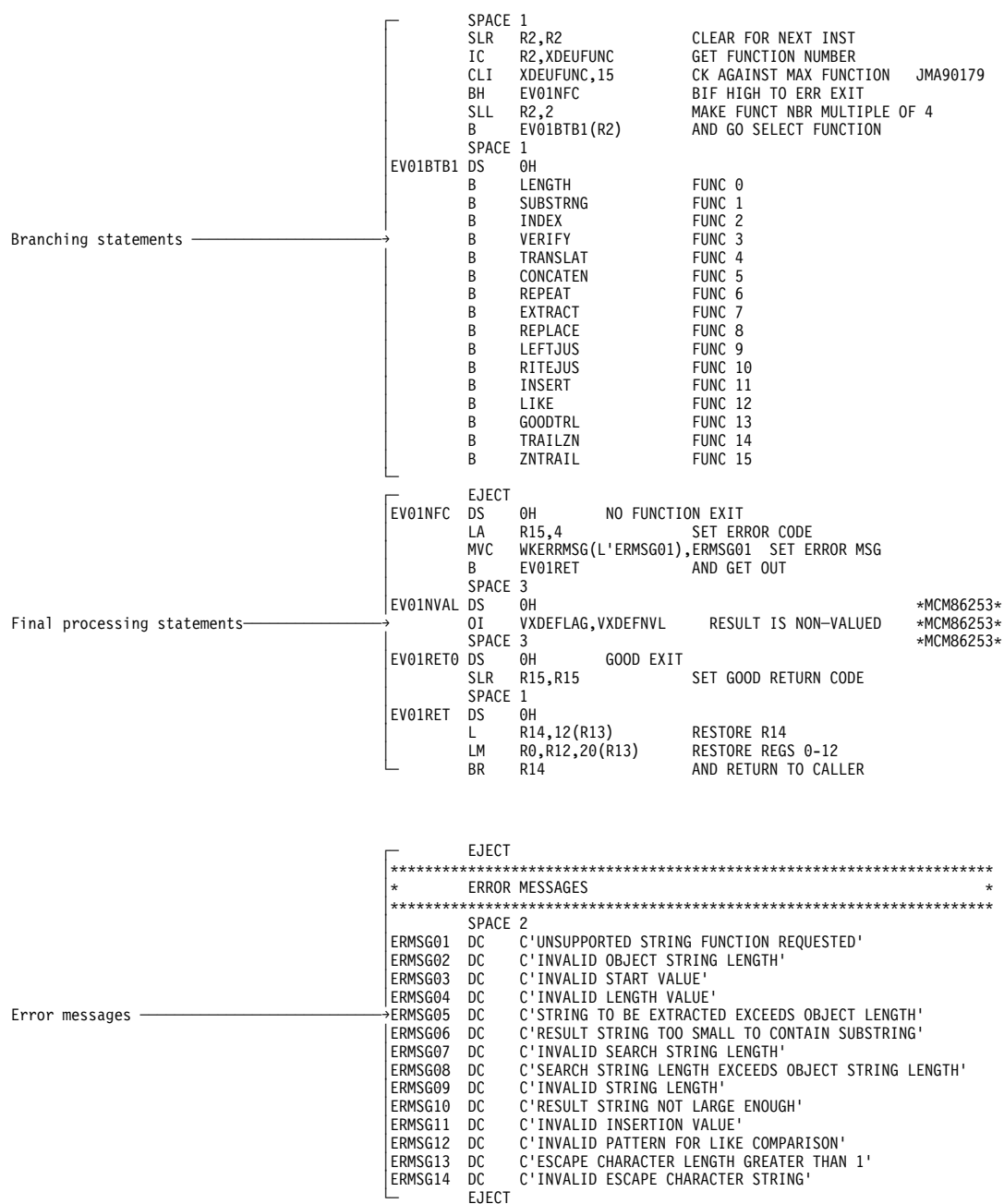
```

RHDCEV01 TITLE 'STRING PROCESSOR FOR RHDCEVAL'
* RHDCEV01 EP=EV01EP1                                06/29/90 14:04:31
*CONTAINS PTF# 90-05-1133                                EXG 05/31/90
*CONTAINS PTF# 88-07-1081                                JMA 02/26/90
*CONTAINS PTF# 87-06-1031                                MCM 08/21/87
*CONTAINS PTF# 85-08-S004                                MCM 03/31/86
* CONTAINS PTF # LEFT/RITE JUST SPA CRM 14:37:29 01/14/85
* CONTAINS PTF # 84-11-1067 CRM 13:37:25 12/14/84
  SPACE 1
  #MOPT CSECT=RHDCEV01,ENV=USER
  SPACE 3

*****
*
* RHDCEV01 IS THE STRING PROCESSOR FOR RHDCEVAL. ALL
* EVAL STRING-HANDLING FUNCTIONS ARE CONTAINED HEREIN.
*
* THESE FUNCTIONS ARE:
*   LENGTH - RETURN LENGTH OF A CHARACTER STRING
*   SUBSTRING - RETURN A SUBSET OF A STRING
*   INDEX - FIND POSITION OF A SUBSTRING
*   VERIFY - INSURE ONE STRING CONTAINS ANOTHER
*   REPLACE - TRANSLATE CHARACTERS
*   CONCATENATE - SHOVE TWO OR MORE STRINGS TOGETHER
*   LIKE - STRING PATTERN MATCHING
*
* UPON ENTRY, R1 MUST CONTAIN THE ADDRESS OF THE OPERATION
* VXDE, WHICH IS BACK-CHAINED TO ALL OPERAND VXDE'S.
*
* ALL STRING INPUT AND OUTPUT WILL BE VARYING-CHARACTER.
* ALL NUMERIC INPUT AND OUTPUT WILL BE HALFWORD-BINARY.
*
*****
EJECT
RHDCEV01 CSECT
RHDCEV01 AMODE ANY
RHDCEV01 RMODE 24
  USING EV01EP1,R12 PROGRAM BASE
  USING EV01EP1+4096,R10 USE SECOND BASE REGISTER
  USING WORKAREA,R11 WORKAREA BASE
  USING XDE,R8
  USING VXDE,R7
  SPACE 1
  ENTRY EV01EP1
  DS 0H
  STM R14,R12,12(R13) SAVE REGISTERS
  LR R12,R15 SET PROGRAM BASE
  L R10,BASE SET UP SECOND BASE REGISTER
  B EV01STRT AND GO START UP
BASE DC A(EV01EP1+4096)
EV01STRT DS 0H
  LR R7,R1 GET RESULT VXDE ADDR
  L R8,VXDEXDEA AND XDE ADDR
  L R11,VXDEUWKA GET WORKAREA ADDR
  STM R7,R8,WKRESADR AND SAVE THEM
  SPACE 1
  MVI WKERRMSG,C' ' NOW BLANK OUT
  MVC WKERRMSG+1(L'WKERRMSG-1),WKERRMSG ERROR MSG FIELD

```

Initialization statements →



F.2 Internal structure of built-in functions

```

*****
*
*      LENGTH - STRING FUNCTION TO RETURN THE LENGTH OF
*      A VARYING-CHARACTER FIELD.
*
*      ONLY REQUIRES ONE OPERAND, THE VARYING-CHAR FIELD.
*      THE RESULT FIELD MUST BE HALFWORD-BINARY.
*
*****
*
*      SPACE 2
*      DS      0H
LENGTH  L      R5,VXDESNXT      GET ADDR OF OPERAND VXDE
        L      R6,VXDEXDEA-VXDE(,R5) AND OPERAND XDE      *MCM86253*
        BAL    R14,CHKNOVAL      *MCM86254*
        LTR    R15,R15           *MCM86254*
        BNZ    EV01INVAL        *MCM86254*
        L      R4,VXDEDADR-VXDE(,R5) GET ADDR OF VC FLD
        MVC    WKFULL(2),0(R4)   MOVE HALFWORD TO ALIGN
        LH     R4,WKFULL         GET LENGTH OF FIELD
        L      R5,VXDEDADR      GET ADDR OF RESULT FLD
        STCM   R4,3,0(R5)       SET ANSWER -STCM FOR BS2K*MCM86090*
        B      EV01RET0         USE GOOD EXIT
        LTORG
        EJECT
*****
*
*      SUBSTRING - STRING FUNCTION TO RETURN A SPECIFIED
*      SUBSET OF A GIVEN STRING
*
*      THIS FUNCTION REQUIRES 3 OPERANDS -
*      1  OBJECT STRING (VARYING-CHARACTER)
*      2  START DISPLACEMENT (HALFWORD)
*      3  LENGTH (OPTIONAL) (HALFWORD)
*      THE RESULT FIELD MUST BE VARYING-CHARACTER ALSO.
*
*      THE OBJECT STRING MAY NOT BE LENGTH ZERO.
*      IF AN ERROR IS DETECTED, THE RESULT STRING LENGTH
*      IS SET TO ZERO, AND AN ERROR RETURNED TO RHDCEVAL.
*
*      REQUIREMENTS OF THE OPERANDS ARE:
*      K = OBJECT STRING LENGTH, I = START DISPLACEMENT,
*      J = LENGTH
*
*      0 LE J LE K          1 LE I LE K
*      I+J-1 LE K
*
*      IF J IS NOT GIVEN, J = K-I+1
*
*      THE OMISSION OF J (3RD OPERAND - LENGTH) IS INDICATED
*      BY A NON-VALUED XDE.
*
*****
*      SPACE 3
*      SUBSTRNG DS      0H
*      NOTE : SUBSTRING OP3 IS AN OPTIONAL PARAMETER.
4*

```

Processing program →

```

*      MUST DIFFERENTIATE BETWEEN OP3 OMITTED          4*
*      AND OP3 SPECIFIED BUT NON-VALUED.                4*
L      R5,VXDES NXT          BACK UP TO OP3 VXDE
L      R6,VXDEXDEA-VXDE(,R5) AND XDE
TM      VXDEFLAG-VXDE(R5),VXDEFNVL IF OP3 VXDE NON-VA *MCM86260*
B0      EV01INVAL          THEN SO IS RESULT *MCM86260*
TM      XDEFLAG-XDE(R6),XDEFNVL IF OP3 XDE NON-VAL *MCM86260*
B0      SUBS0010          THEN CHK FURTHER *MCM86260*
B       SUBS0040          CONTINUE WITH OP2 *MCM86260*
SPACE 1 *MCM86260*
SUBS0010 DS 0H *MCM86260*
CLC      XDEDATAD-XDE(,R6),=X'80000000' OP3 OMITTED? *MCM86260*
BE      SUBS0040          YES - CONTINUE *MCM86260*
B       EV01INVAL          NO - OP3 NON-VALUED *MCM86260*
SPACE 1 *MCM86260*
SUBS0040 DS 0H *MCM86260*
STM      R5,R6,WKOP3SV          SAVE OP3 XDE,VXDE *MCM86260*
L      R5,VXDES NXT-VXDE(,R5) BACK UP TO OP2 VXDE
L      R6,VXDEXDEA-VXDE(,R5) AND XDE
BAL      R14,CHKNOVAL
*MCM86254*
LTR      R15,R15
*MCM86254*
BNZ      EV01INVAL
*MCM86254*
STM      R5,R6,WKOP2SV          SAVE OP2 XDE,VXDE
*MCM86253*
SPACE 1
L      R5,VXDES NXT-VXDE(,R5) BACK UP TO OP1 VXDE
L      R6,VXDEXDEA-VXDE(,R5) AND XDE
BAL      R14,CHKNOVAL
*MCM86254*
LTR      R15,R15
*MCM86254*
BNZ      EV01INVAL
*MCM86254*
STM      R5,R6,WKOP1SV          SAVE OP1 XDE,VXDE
*MCM86253*
EJECT
*****
Processing program (cont'd) *****
OBJECT STRING LENGTH MUST BE GREATER THAN ZERO *****
L      R4,VXDEDADR-VXDE(,R5) GET OP1 DATA ADDR
MVC      WKFULL,0(R4)          GET HALFWORD LNG FROM VC FLD
LH      R4,WKFULL          PUT INTO A REGISTER
LTR      R4,R4          CK IT FOR ZERO
BP      SUBS0050          GTR ZERO IS OKAY - BRANCH
SPACE 1
MVC      WKERRMSG(L'ERMSG02),ERMSG02 SET ERROR MSG
B       SUBS0950          USE ERROR EXIT
SPACE 1
SUBS005 DS 0H          R4 NOW CONTAINS LENGTH OF OBJECT STRING
***** CHECK STARTING DISPLACEMENT *****
LM      R7,R8,WKOP2SV          GET VXDE/XDE ADDRS, OP2
L      R3,VXDEDADR          GET OP2 DATA ADDR
MVC      WKFULL,0(R3)          GET HALFWORD DATA FIELD
LH      R3,WKFULL          GET THE VALUE
LTR      R3,R3          CK FOR ZERO OR LESS
BP      SUBS0080          IF POSITIVE, BRANCH
SPACE 1

```

F.2 Internal structure of built-in functions

Processing program (cont'd)

```

SUBS0075 DS    0H          INVALID START FIELD
          MVC    WKERRMSG(L'ERMSG03),ERMSG03  SET ERROR MSG
          B      SUBS0950          USE ERROR EXIT
          SPACE 1
SUBS0080 DS    0H
          CR     R3,R4            COMPARE TO MAX START
          BH     SUBS0075          ERR IF START GTR LENGTH
          SPACE 1
***** R3 NOW HAS STARTING DISPLACEMENT RELATIVE TO ONE *****
***** NOW GET EXTRACT LENGTH, WHICH MIGHT HAVE BEEN OMITTED ***
          LM     R7,R8,WKOP3SV    GET VXDE/XDE ADDRS
          TM     XDEFLAG,XDEFNVL  CK FOR PARMETER OMITTED
          BZ     SUBS0090          BIF IT IS PRESENT
          SPACE 1
SUBS0085 DS    0H
          LR     R2,R4            ELSE SET
          SR     R2,R3            EXTRACT LNG TO
          LA     R2,1(R2)          TOTAL-START+1
          B      SUBS0120          AND BYPASS NEXT EDIT
          SPACE 1
SUBS0090 DS    0H
          L      R2,VXDEDADR      GET DATA ADDR, OP3
          MVC    WKFULL,0(R2)     GET HALFWORD LENGTH
          LH     R2,WKFULL        PUT INTO A REGISTER
          LTR    R2,R2            CK FOR ZERO OR LESS
          BZ     SUBS0085          ZERO - TAKE DEFAULT ABOVE
          BP     SUBS0100          POSITIVE IS OKAY - BRANCH
          SPACE 1
SUBS0095 DS    0H          INVALID LENGTH FIELD
          MVC    WKERRMSG(L'ERMSG04),ERMSG04  SET ERROR MESSAGE
          B      SUBS0950          USE ERROR EXIT
          SPACE 1
SUBS0100 DS    0H
          CR     R2,R4            MUST BE LESS THAN OBJECT-LNG
          BH     SUBS0095          IF NOT, ERROR
          EJECT
SUBS0120 DS    0H
***** INSURE START + EXTRACT LNG DOESN'T EXCEED OBJECT STRING
LNG**
          LR     R1,R3            GET START DISPLACEMENT
          AR     R1,R2            ADD EXTRACT LENGTH
          BCTR   R1,0             DECREMENT BY ONE
          CR     R1,R4            COMPARE TO TOTAL AVAIL
          BNH    SUBS0140          EQ OR LOW IS OKAY - BRANCH
          SPACE 1
          MVC    WKERRMSG(L'ERMSG05),ERMSG05  SET ERROR MESSAGE
          B      SUBS0950          USE ERROR EXIT
          SPACE 1

```

Processing program (cont'd)

```

SUBS0140 DS    0H
***** R4 HAS TOTAL STRING LENGTH OF OBJECT *****
***** R3 HAS DISPLACEMENT TO START OF EXTRACT *****
***** R2 HAS LENGTH TO EXTRACT *****
***** MUST NOW TEST RESULT FIELD SIZE TO INSURE IT CAN *****
***** CONTAIN THE EXTRACTED SUBSTRING *****
SPACE 1
LM    R7,R8,WKRESADR    GET VXDE/XDE ADDRS OF RESULT
LH    R5,XDEDATLN       GET MAX RESULT SIZE
CR    R2,R5             EXTRACT LNG CAN'T EXCEED TARG LEN
BNH   SUBS0150          BRANCH IF OKAY
SPACE 1
MVC   WKERRMSG(L'ERMSG06),ERMSG06 SET ERROR MESSAGE
B     SUBS0950          USE ERROR EXIT
SUBS0150 DS    0H      NOW READY TO EXTRACT THE SUBSTRING
L     R5,VXDEDADR       GET RESULT FLD ADDR
STH   R2,WKFULL         ALIGN THE SUBSTRING LENGTH
MVC   0(2,R5),WKFULL    PUT LENGTH FIELD INTO RESULT (VC)
LA    R5,2(,R5)         AND ADVANCE RESULT FIELD POINTER
SPACE 1
L     R6,WKOP1SV        GET VXDE ADDR OBJECT STRING
L     R6,VXDEDADR-VXDE(,R6) GET DATA ADDR
LA    R6,2(,R6)         BUMP PAST LENGTH FIELD
BCTR  R3,0             MAKE START RELATIVE TO ZERO
AR    R6,R3            CALC ADDR OF SUBSTRING
SPACE 1
LR    R4,R2            GET LENGTH IN RIGHT REGISTER
BAL   R14,MOVEIT       MOVE THE SUBSTRING
B     SUBS0980          AND USE SUCCESS EXIT
EJECT
SUBS0950 DS    0H      ERROR EXIT
LM    R7,R8,WKRESADR    GET RESULT VXDE/XDE ADDRS
L     R6,VXDEDADR       GET DATA FIELD ADDR
XC    0(2,R6),0(R6)     SET LNG TO NULL
LA    R15,4            SET ERROR RETURN CODE
B     EV01RET          AND USE ERROR EXIT
SPACE 2
SUBS0980 DS    0H      SUCCESS EXIT
B     EV01RET0         USE GOOD EXIT
LTORG
EJECT

```

```

*****
*
*
*
*      WORKAREA  —  PASSED BY CALLER
*
*
*****
*
*      SPACE 2
WORKAREA DSECT
WKERRMSG DS      CL80
WKFULL   DS      F
          ORG     WKFULL
WKHALF   DS      H      *THIS CAN'T BE USED WITH WKFULL !!
WKFLAG1  DS      CL1    WORK FLAG1
WKFLASTR EQU     X'80'   GOT ME AN ARBITRARY STRING WORKING
WKFIOP30 EQU     X'40'   ESCAPE CHAR IN REQUEST
WKFIOP3S EQU     X'20'   ESCAPE CHAR ENCOUNTERED
WKFIEX10 EQU     X'10'   X'10' BIT FOR FLAG1
WKFIPON  EQU     X'08'   PROCESSING % OPERATOR
WKFIPOSET EQU     X'04'  1ST CHAR IN % STR MATCHED IN OBJ
Work area storage definition -> WKFIEX02 EQU     X'02'  X'02' BIT FOR FLAG1
WKFIEX01 EQU     X'01'  X'01' BIT FOR FLAG1
WKFLAG2  DS      CL1    WORK FLAG2
WKRESADR DS      2F     VXDE/XDE ADDRS, RESULT FIELD
WKRESLADR DS      F     RESULT ADDR
WKRESLLNG DS      F     RESULT LENGTH
WKOP1SV  DS      2F     VXDE/XDE ADDRS, OPERAND 1 FIELD
WKOP2SV  DS      2F     VXDE/XDE ADDRS, OPERAND 2 FIELD
          ORG     WKOP2SV
WKPATCNT DS      F     STARTING COUNT FOR % PATTERN
WKPATADR DS      F     STARTING POSITION FOR % PATTERN
WKOP3SV  DS      2F     VXDE/XDE ADDRS, OPERAND 3 FIELD
          ORG     WKOP3SV
WKOBJCNT DS      F     STARTING COUNT FOR % PATTERN
WKOBJADR DS      F     STARTING POSITION FOR % PATTERN
WKOP1LNG DS      F     OPERAND 1 LENGTH
WKOP2LNG DS      F     OPERAND 2 LENGTH
WKOP3LNG DS      F     OPERAND 3 LENGTH
WKOP1ADR DS      F     OPERAND 1 ADDRESS
WKOP2ADR DS      F     OPERAND 2 ADDRESS
WKOP3ADR DS      F     OPERAND 3 ADDRESS
          SPACE 2
WKLENGTH EQU     *-WORKAREA  LENGTH OF WORKAREA
          EJECT
          COPY    #XDEDS
          END     EV01EP1

```

F.2.5 Runtime processing of built-in functions

At runtime, the following processing sequence occurs for each function:

1. The **runtime system** begins processing the function, as follows:

- Moves each parameter in the function to an intermediate result area (IRA). If a parameter is coded as a multi-operand expression, the expression is evaluated and only the result is moved to the IRA. Data conversions are performed as necessary.

The runtime system maintains an operand XDE/VXDE for each function parameter. The XDE/VXDE pair describes the parameter as it is stored in the IRA and contains the parameter's IRA address.

►► For details on processing optional parameters, see F.5.1, “Steps for generating a user-defined built-in function” later in this appendix.

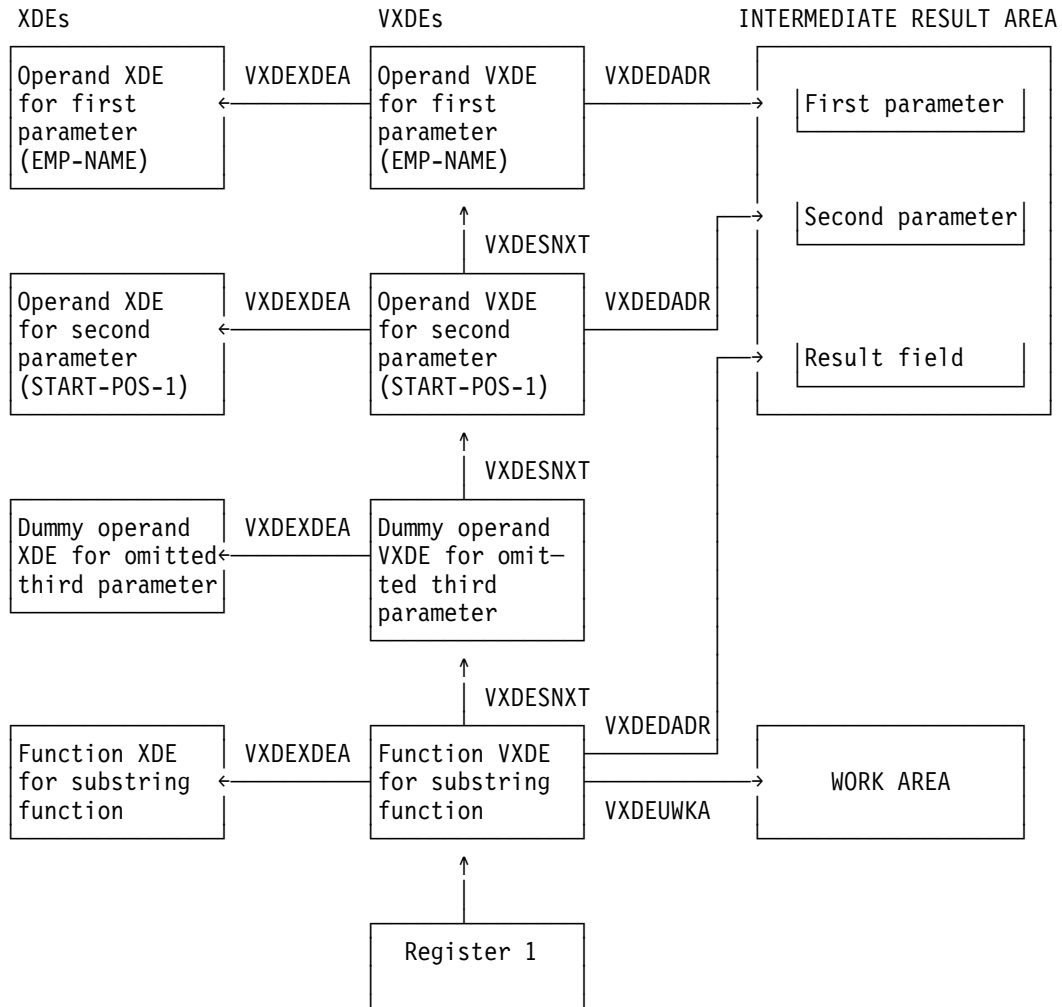
- Passes control to the processing program module named in the function XDE. The runtime system places in register 1 the address of the function VXDE. (A function VXDE contains addresses that enable access to the work area, the result field in the IRA, and the function's operand XDEs/VXDEs.)
2. The **processing program** continues processing the function, as follows:
- Processes initialization statements, as illustrated earlier in this appendix. Register information is saved by using standard OS/390 conventions. Register 1 is used to access the function VXDE, which in turn enables access to all the information required for processing the function, as illustrated below.
 - Branches to the appropriate program by using a branching routine in conjunction with the XDEUFUNC field of the function XDE.
 - Processes the program statements.
 - Processes final statements. If the program did not encounter an error condition, register 15 is set to 0. If an error was encountered, register 15 is set to a nonzero value and an optional error message is moved to the first 80 bytes of the work area. Registers are restored and the module passes control back to the runtime system.
3. The **runtime system** finishes processing the function by checking the value returned in register 15. If register 15 equals 0, the runtime system resumes the process. The result of the function is used in the statement in which the function is coded. If register 15 is greater than 0, the runtime system aborts the dialog and displays the Dialog Abort Information screen along with the optional error message.

Internal representation of a function at run time: The internal representation of a substring function is illustrated below.

An arrow indicates that the source structure contains the address of the object structure, illustrating that the processing program module can use register 1 to gain access to all required information.

Field names containing the addresses are listed next to the arrows.

SUBSTRING(EMP-NAME, START-POS-1)



F.3 Assembler macros

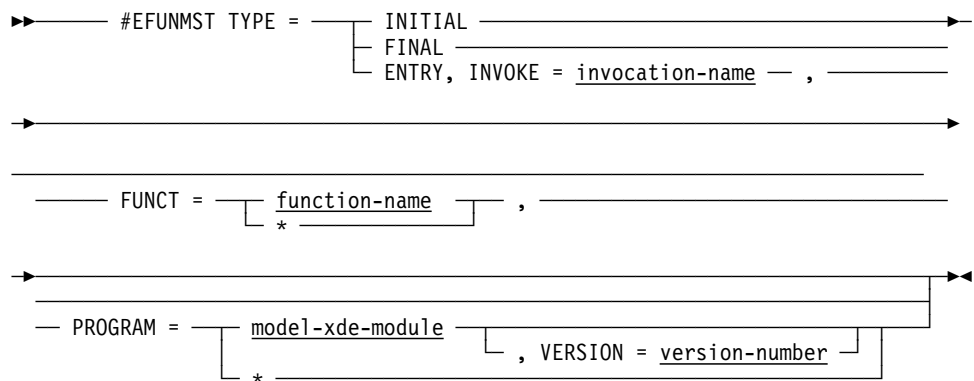
Assembler macros (#EFUNMST and #EFUNMOD) are used in assembler programs to define the master function table and the model XDE modules. The programs are assembled to create object modules. The object module for the master function table is then placed in the data dictionary load area by using the DDDL compiler. The object module for a model XDE module is placed in the load library by using the linkage editor.

The macros #EFUNMST and #EFUNMOD are discussed separately below.

F.3.1 #EFUNMST

Purpose: Defines the master function table. Three types of #EFUNMST macros are coded in a source assembler program, as follows:

Syntax:



Parameters

INITIAL

Generates header information and automatically generates the #EFUNMST TYPE=ENTRY macros for the CA-ADS supplied built-in functions.

TYPE=INITIAL is coded first and only once in the assembler program.

FINAL

Defines the end of the table.

ENTRY

Generates an entry in the master function table.

TYPE=ENTRY macros are coded once for each entry in the table.

INVOKE= invocation-name

Specifies a user-defined, 1- to 32-character invocation name for the function.

FUNCT= function-name/*

Specifies a user-defined, 1- to 8-character real (generic) function name for the function.

This function name is used to associate the coded invocation name with the model XDE table that describes the function in the model XDE module. The character * can be specified if the real function name is the same as the real function name for the previous entry in the master function table.

PROGRAM= model-xde-module

Specifies the 1- to 8-character name of the model XDE module in which the function is described.

,VERSION= version-number

Specifies the 1- to 4-digit version number of the model XDE module in which the function is described.

*

The character * can be specified if the model XDE module name is the same as the model XDE module name for the previous entry in the master function table (see RHDCEVBF below).

Usage:

Considerations

- A source module must begin with one TYPE=INITIAL macro and end with one TYPE=FINAL macro.
- Any number of TYPE=ENTRY macros can be coded between the INITIAL and FINAL type macros.

F.3.2 RHDCEVBF

The master function table is defined in a source assembler program called RHDCEVBF. RHDCEVBF is shown below as it appears when CA-ADS is installed. Entries for user-defined functions are defined by coding #EFUNMST TYPE=ENTRY macros between the INITIAL and FINAL type macros.

►► For more information, see F.4, “Changing invocation names” later in this appendix.

Source assembler program RHDCEVBF

```
RHDCEVBF TITLE 'EVAL - BUILT-IN FUNCTIONS - MASTER TABLE'
* RHDCEVBF EP=RHDCEVBF                                06/25/90 14:52:50
    #EFUNMST TYPE=INITIAL                               12/08/88 15:52:14
    EJECT
    #EFUNMST TYPE=FINAL
    END
```

The TYPE=INITIAL macro automatically generates the entries for the CA-ADS supplied built-in functions. It does this by copying the TYPE=ENTRY macros coded in the source module #EFMBIFS. A segment of source module #EFMBIFS is shown below. Invocation names for the CA-ADS supplied built-in functions can be changed

by modifying the source module #EFMBIFS, as described under 'Changing Invocation Names' later in this appendix.

Segment of source module #EFMBIFS

```

*      #EFMBIFS  EVAL BUILT-IN FUNCTIONS - MASTER DEFS
***** FUNCTION = LENGTH  (STRING FUNCTION) *****
SPACE 2
**** INVOCATION NAME = SLENGTH ****
      #EFUNMST TYPE=ENTRY,
      INVOKE=SLENGTH,
      FUNCT=LENGTH,
      PROGRAM=RHDCEV51
SPACE 2
**** INVOCATION NAME = STRING-LENGTH ****
      #EFUNMST TYPE=ENTRY,
      INVOKE=STRING-LENGTH,
      FUNCT=*,
      PROGRAM=*
SPACE 2
**** INVOCATION NAME = SLEN ****
      #EFUNMST TYPE=ENTRY,
      INVOKE=SLEN,
      FUNCT=*,
      PROGRAM=*
EJECT
***** FUNCTION = SUBSTRING (STRING FUNCTION) *****
SPACE 2
**** INVOCATION NAME = SUBSTRING ****
      #EFUNMST TYPE=ENTRY,
      INVOKE=SUBSTRING,
      FUNCT=SUBSTRNG,
      PROGRAM=RHDCEV51
SPACE 2
**** INVOCATION NAME = SUB-STRING ****
      #EFUNMST TYPE=ENTRY,
      INVOKE=SUBSTR,
      FUNCT=*,
      PROGRAM=*
SPACE 2
**** INVOCATION NAME = SUBS ****
      #EFUNMST TYPE=ENTRY,
      INVOKE=SUBS,
      FUNCT=*,
      PROGRAM=*
EJECT
***** FUNCTION = INDEX  (STRING FUNCTION) *****
SPACE 2
**** INVOCATION NAME = INDEX ****
      #EFUNMST TYPE=ENTRY,
      INVOKE=INDEX,
      FUNCT=INDEX,
      PROGRAM=RHDCEV51
SPACE 2
**** INVOCATION NAME = STRING-INDEX ****
      #EFUNMST TYPE=ENTRY,
      INVOKE=STRING-INDEX,
      FUNCT=*,
      PROGRAM=*
SPACE 2
**** INVOCATION NAME = INDX ****
      #EFUNMST TYPE=ENTRY,
      INVOKE=INDX,
      FUNCT=*,
      PROGRAM=*

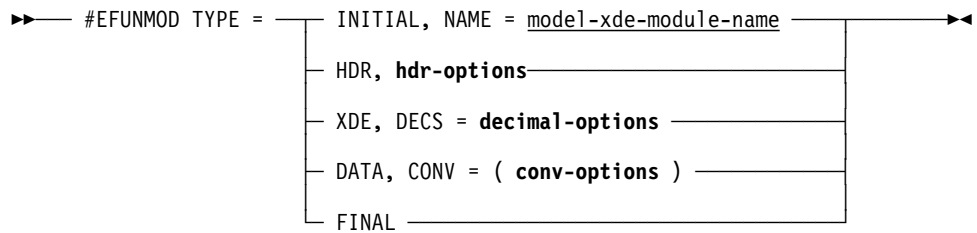
```

F.3.3 #EFUNMOD

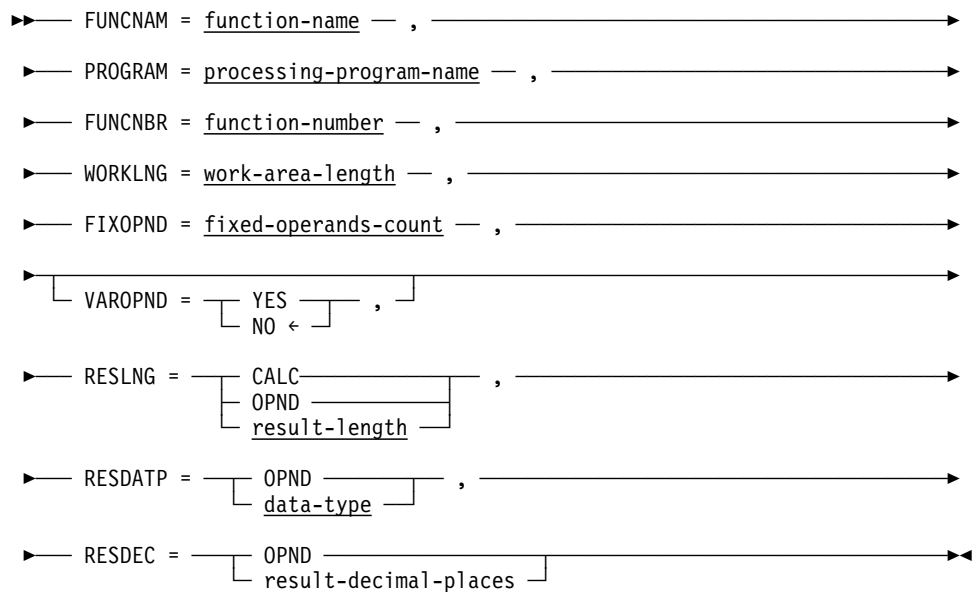
Purpose: Defines a model XDE module and the model XDE tables within the module.

Each model XDE table describes one function. During process compilation of a built-in function, the dialog compiler uses the appropriate model XDE table to convert the built-in function into a series of XDEs, which represents the function at runtime.

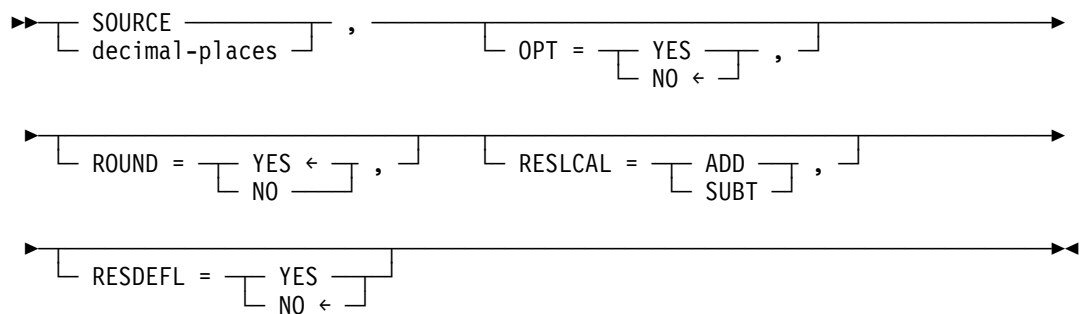
Syntax:

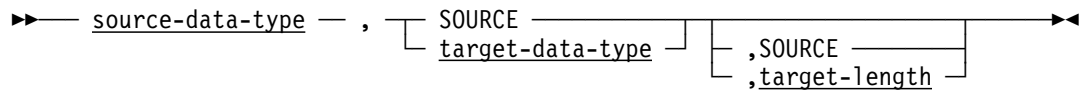


Expansion of hdr-options



Expansion of decimal-options



Expansion of source-specification**Parameters****INITIAL, NAME = model-xde-module-name**

Specifies the 1- to 8-character model XDE module name.

The TYPE=INITIAL macro is coded first and only once in the assembler program.

HDR, hdr-options

Defines the beginning of a model XDE table and specifies function XDE information.

One TYPE=HDR macro is coded for each model XDE table.

See expansion of hdr-options below.

XDE, DECS = decimal-options

Specifies operand XDE information that describes a target parameter.

The TYPE=XDE macro describes a function parameter. One TYPE=XDE macro is coded for each parameter in the order that the parameter is to appear in the parameter list.

DECS = decimal-options is used to specify the number of decimal places in the target parameter being described.

See the expansion of decimal-options below.

DATA, CONV = (conv-options)

Specifies the data type and length of the target parameter (that is, the parameter as it is stored in the IRA for use by the processing program), based on the data type of the source parameter (that is, the parameter as it is coded in the parameter list).

See expansion of conv-options below.

At least one TYPE=DATA macro must be coded following a TYPE=XDE macro. If two or more are specified, the dialog compiler uses the TYPE=DATA macro whose *source-data-type* specification matches the data type of the source parameter. If no *source-data-type* specification matches, the last TYPE=DATA macro is used.

Note that during process compilation, any combination of source and target parameter data types is accepted. At runtime, the runtime system attempts to make any required data type conversions; if it cannot, the dialog aborts.

➡ For more information about data type conversion, see Chapter 5, “Introduction to Process Language.”

FINAL

Defines the end of the model XDE module.

Expansion of hdr-options**FUNCNAM= function-name**

User-defined parameter specifying the 1- to 8-character real (generic) function name.

The real function name associates a master function table entry with the model XDE table.

PROGRAM= processing-program-name

User-defined parameters specifying the 1- to 8-character name of the processing program module that contains the processing program for the function.

FUNCNBR= function-number

User-supplied numeric literal specifying a number from 0 to 255 that uniquely identifies the associated processing program within the processing program module.

WORKLNG= work-area-length

User-supplied numeric literal specifying the number of bytes of work area required by the processing program module for the function.

The WORKLNG specification should not include work space required by the runtime system, which is automatically added by the macro.

Note: *Work-area-length* must be at least 80.

FIXOPND= fixed-operands-count

User-supplied numeric literal specifying the number of fixed parameters for the function.

A fixed parameter is a parameter that can be specified only once in a parameter list. A function can have from 0 to 50 fixed parameters.

VAROPND=YES/NO

Specifies whether one parameter in the parameter list is variable.

A variable parameter can be specified repeatedly in a parameter list. (An example of a variable parameter is *'string'* or *string-variable* in the concatenate function.)

A function can have only one variable parameter and it must follow all fixed parameters.

The default VAROPND specification is NO.

RESLNG=

Clause introducing the length, in bytes, of the function's result field,

CALC

Specifies that the result field length is calculated from the lengths of the function parameters, based on the RESLNG specification of each parameter's TYPE=XDE macro.

If CALC is specified, the result field length is calculated as the sum of the lengths of the function parameters whose RESLNG specification is ADD, minus the sum of the lengths of the function parameters whose RESLNG

specification is SUBT. Parameters without a RESLNG specification are not included in the calculation.

OPND

Specifies that the result length is equal to the length of the function parameter whose RESDEFL specification is YES.

result-length

Specifies a result length, in bytes, from 1 to 32767.

RESDATP=

Clause introducing the data type of the function's result field of the function.

OPND

Specifies that the result data type is the same as the data type of the function parameter whose RESDEFL specification is YES.

data-type

User-defined parameter specifying the result field data type.

Data-type is one of the three-character data type abbreviations shown in the table under **Usage** below.

RESDEC=

Clause introducing the number of decimal places in the function's result field.

OPND

Specifies that the number of decimal places is equal to the number of decimal places in the function parameter whose RESDEFL specification is YES.

result-decimal-places

Specifies the number of result decimal places, from 0 to 32.

Expansion of decimal-options**SOURCE**

Specifies that the number of decimal places equals the number of decimal places in the source parameter.

decimal-places

Specifies the number of decimal places, from 0 to 32.

OPT=YES/NO

Specifies whether the parameter is optional and can be omitted from the coded parameter list.

NO is the default when neither YES or NO is specified.

ROUND=YES/NO

Specifies whether rounding or truncation is used when converting from the source parameter to the target parameter. NO indicates truncation.

YES is the default when neither YES or NO is specified.

RESLCAL=

Clause introducing the action to be taken to the parameters length in the calculation of the length of the result field.

ADD

Specifies that the parameter's length is added in the calculation of the length of the result field.

SUBT

Specifies that the parameter's length is subtracted in the calculation of the length of the result field.

The RESLCAL specification should be included only if the RESLNG specification of the preceding TYPE=HDR macro is CALC. If the RESLCAL specification is omitted, the parameter's length is not considered in the calculation of the length of the result field.

RESDEFL=YES/NO

Specifies whether the parameter is used to determine result field characteristics that are specified in the associated TYPE=HDR macro as OPND.

Only one TYPE=XDE macro for a function can specify RESDEFL=YES.

The default RESDEFL specification is NO.

Expansion of conv-options**source-data-type**

User-defined parameters specifying the three-character abbreviation of the data type of the source parameter; these abbreviations are listed in the table under Usage below.

During process compilation, if the data type of the source parameter is *source-data-type*, then the target parameter is assigned a data type of *SOURCE/target-data-type* and a length of *SOURCE/target-length*. The target parameter's data type and length are stored in the parameter's operand XDE.

SOURCE

Specifies that the data type of the target parameter is the same as the data type of the source parameter.

target-data-type

User-defined parameters specifying the three-character abbreviation of the data type of the target parameter; these abbreviations are listed in the table under Usage below.

SOURCE

Specifies that the length of the target parameters is the same as the length of the source parameter.

target-length

Specifies the length of the target parameter in bytes.

If neither is specified, a length is generated based on the data type of the target parameter, if possible.

Usage:

Considerations

- The source assembler program must begin with one TYPE=INITIAL macro and end with one TYPE=FINAL macro.
- One TYPE=HDR macro is coded for each function that is described in the module.
- One TYPE=XDE macro is coded for each parameter of each function; the macro applies to the function described by the preceding TYPE=HDR macro and is coded in the order that the parameter is to appear in the parameter list.
- One or more TYPE=DATA macros are coded for each data type conversion specification for each parameter; the macro applies to the parameter described by the preceding TYPE=XDE macro.

Data type abbreviations

Data type	Abbreviation
Display floating point	DFL
Doubleword binary	DWB
EBCDIC	EBD
Fullword binary	FWB
Group	GRP
Halfword binary	HWB
Long floating point	LFL
Multibit binary	MBB
Short floating point	SFL
Signed packed decimal	SPK
Signed zoned decimal	SZN
Unsigned packed decimal	UPK
Unsigned zoned decimal	UZN
Varying character	VCH

Note: Only target parameters and the result field can have the varying character data type. A varying character field consists of a halfword binary field that specifies the length of the varying character string, followed by a fixed field that contains the string itself.

Model XDE modules: The model XDE modules for the CA-ADS supplied built-in functions are defined by the source assembler programs called RHDCEV51, RHDCEV52, and RHDCEV53. Segments of RHDCEV51 are shown below. An

installation should not change these modules, but can reference them as guides for creating user-defined built-in functions.

►► For more information, F.5, “Creating user-defined built-in functions” later in this appendix.

Segments of source assembler program RHDCEV51

```
RHDCEV51 TITLE 'EVAL - BUILT-IN STRING FUNCTIONS - MODEL XDE TBL'
* RHDCEV51 EP=RHDCEV51                                06/29/90 14:05:40
    SPACE 3
RHDCEV51 AMODE ANY
RHDCEV51 RMODE 24
    #EFUNMOD TYPE=INITIAL,NAME=RHDCEV51
EJECT
*****
*      FUNCTION = LENGTH                                *
*****
    SPACE 3
LENGTH  #EFUNMOD TYPE=HDR,                                X
        FUNCNAM=LENGTH,                                X
        FUNCNBR=0,                                    X
        PROGRAM=RHDCEV01,                              X
        WORKLNG=148,                                  X
        FIXOPND=1,                                    X
        RESLNG=2,                                      X
        RESDATP=HWB,                                  X
        RESDEC=0
    SPACE 1
    #EFUNMOD TYPE=XDE,DECS=0,OPT=NO
    SPACE 1
    #EFUNMOD TYPE=DATA,CONV=(EBD,VCH,SOURCE)
EJECT
```

```

*****
*      FUNCTION = SUBSTRING      *
*****
      SPACE 3
SUBSTRNG #EFUNMOD TYPE=HDR,          X
          FUNCNAM=SUBSTRNG,          X
          FUNCNBR=1,                 X
          PROGRAM=RHDCEV01,          X
          WORKLNG=148,               X
          FIXOPND=3,                 X
          RESLNG=CALC,               X
          RESDATP=VCH,               X
          RESDEC=0
      SPACE 1
      #EFUNMOD TYPE=XDE,DECS=0,OPT=NO,RESLCAL=ADD
      SPACE 1
      #EFUNMOD TYPE=DATA,CONV=(EBD,VCH,SOURCE)
      SPACE 1
      #EFUNMOD TYPE=XDE,DECS=0,OPT=NO
      SPACE 1
      #EFUNMOD TYPE=DATA,CONV=(EBD,HWB,2)
      SPACE 1
      #EFUNMOD TYPE=XDE,DECS=0,OPT=YES
      SPACE 1
      #EFUNMOD TYPE=DATA,CONV=(EBD,HWB,2)
      EJECT
...
      #EFUNMOD TYPE=FINAL
      SPACE 2
      END    RHDCEV51                *CRM84199*

```

F.4 Changing invocation names

An installation can add, modify, or delete any invocation name for any CA-ADS supplied or user-defined built-in function. The following steps update the master function table, which contains the valid invocation names:

1. **Modify source macro #EFMBIFS** — #EFMBIFS contains the assembler macros that define the entries in the master function table for the CA-ADS supplied built-in functions. Invocation names can be changed by adding, modifying, and/or deleting the appropriate #EFUNMST macros in #EFMBIFS, then following the steps listed below. Refer to the syntax rules for the #EFUNMST macro earlier in this appendix.
2. **Modify source module RHDCEVBF** — RHDCEVBF contains the #EFUNMST assembler macros that define the master function table, including the TYPE=INITIAL macro, which automatically generates the macros stored in #EFMBIFS.

Invocation names for user-defined functions are defined by TYPE=ENTRY macros coded between the TYPE=INITIAL and TYPE=FINAL macros in RHDCEVBF. TYPE=ENTRY macros can be added, modified, and deleted as required.

RHDCEVBF also contains PUNCH statements that prefix the module with the required IDD statement to place the master function table in the data dictionary load area. Change the action ADD to MOD if it has not already been changed.

3. **Assemble source module RHDCEVBF** — The object module generated should also be called RHDCEVBF.
4. **Place RHDCEVBF in the data dictionary load area** — Use the DDDL compiler.

►► JCL for the DDDL compiler is presented in *IDD DDDL Reference*.

F.5 Creating user-defined built-in functions

Built-in functions can be created to meet site-specific needs. User-defined built-in functions are coded like the CA-ADS supplied functions.

The following topics are discussed below:

- Steps for generating a user-defined built-in function
- LRF Considerations
- Calling a user-defined built-in function

F.5.1 Steps for generating a user-defined built-in function

An installation can generate a user-defined function by following the instructions listed below in any order:

- **Create a processing program module**, as follows:

1. **Create the source module** — As a guide, refer to the source module RHDCEV01 which contains some of the source code for the CA-ADS supplied built-in functions.

Processing logic for several functions can be included in one processing program module, thereby reducing the number of modules that must be loaded at runtime. Each function is distinguished by a unique function number. The function number is defined in the model XDE module by the FUNCNBR parameter of the #EFUNMOD TYPE=HDR macro. At runtime, the function number is contained in the XDEUFUNC field of the function XDE, and can be used by the processing program module to branch to the appropriate processing program.

Note: A built-in function that supports entry of optional parameters must be able to determine at execution time whether the optional parameters have been entered. To do this, the built-in function must perform a runtime check of the XDE/VXDE for each such parameter.

If an optional parameter is omitted, the runtime system passes a dummy operand VDE/VXDE to the built-in function for the omitted parameter. The dummy XDE/VXDE for the parameter has the following characteristics, for which a built-in function can test:

- The XDEFNVL bit in the XDEFLAG field is set to 1.
- The XDEDATAD field is set to X'80000000'.

2. **Assemble the processing program source module.**
3. **Link edit the module into the load library.**
4. **Add the program at system generation with the PROGRAM statement.**
 - For information about linking built-in functions with the runtime system, see Chapter 4, “CA-ADS Runtime System.”

- **Create a model XDE module**, as follows:
 1. **Create the source module** — The source module consists of #EFUNMOD macros. As a guide, refer to RHDCEV51, RHDCEV52, RHDCEV53, RHDCEV59, and RHDCEV60, the model XDE source modules for the CA-ADS supplied built-in functions.
 2. **Assemble the source module.**
 3. **Link edit the module into the load library.**
- **Update the master function table** by following the steps described under F.4, “Changing invocation names” earlier in this appendix.

F.5.2 LRF considerations for user-defined built-in functions

If a site-defined built-in function is used with the Logical Record Facility (LRF) WHERE clause, the function must check each parameter to determine if the record containing the parameter value has been read by LRF processing. If the value has been read, the parameter is considered *valued*. If the value has not yet been read, the parameter is *nonvalued*.

A parameter is checked for being nonvalued by examining its associated XDE and VXDE. The exact checks that need to be made depend on whether a parameter is optional or required for that particular built-in function. The following considerations apply:

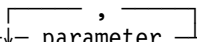
- **If a parameter is optional**, it is nonvalued if VXDEFNVL is ON, or if XDEFNVL is ON and XDEDATAD is not equal to X'80000000'.
- **If a parameter is required**, it is nonvalued if either of its XDEFNVL or VXDEFNVL bits is ON.

If any parameter is nonvalued, the built-in function must react accordingly. The proper action to take depends on the function being performed and which parameter is nonvalued. In most cases, the built-in function will return a nonvalued result by setting the VXDEFNVL flag in the result VXDE.

F.5.3 Calling a user-defined built-in function

Purpose: This is the generalized syntax for calling a user-defined built-in function.

Syntax:

►► invocation-name () ►►

Parameters

invocation-name

Specifies the invocation name for the user-defined function.

parameter

Specifies the parameters for the user-defined function.

Optional parameters that are not included must be replaced by the character @, unless no included parameters follow the omitted parameter.

Appendix G. Security Features

- G.1 Overview G-3
- G.2 CA-ADS compiler security G-4
- G.3 CA-ADS application security G-5
 - G.3.1 Response security G-5
 - G.3.2 Signon security G-6

G.1 Overview

In the CA-IDMS environment, use of the CA-ADS compilers and use of the CA-ADS applications that you develop with the compilers can be secured.

Compiler security: Use of the CA-ADS compilers can be secured through the CA-IDMS central security system at various levels, such as at the task level and at the program level. A dictionary that the compiler accesses can be secured as a database.

►► For more information about CA-IDMS central security, refer to *CA-IDMS Security Administration*.

Use of the CA-ADS compilers to access particular dictionary entities can also be controlled. You can secure access to dictionaries that the CA-ADS compilers access using DDDL statements.

►► For more information about securing CA-ADS compilers using DDDL statements, see G.2, “CA-ADS compiler security” later in this appendix.

Application security: Use of CA-ADS applications can be secured through CA-IDMS central security at various levels, such as at the task level, the program level, and the activity level. Databases that the application accesses, including dictionaries, can be secured.

CA-ADS security classes are used when activities are secured in CA-IDMS. You can also specify a security class for the CA-ADS application and security classes for application responses. At runtime, the application issues a request for a security check when a user tries to execute an application or an application response for which you have specified a security class. If activity security has been enabled, CA-IDMS central security checks to see whether the user has authority to execute the activity whose activity number matches the security class.

►► For more information about CA-IDMS activity security, refer to *CA-IDMS Security Administration*.

For information about specifying a security class in CA-ADS, see G.3.1, “Response security” later in this appendix.

When you define a CA-ADS application, you can specify that the user must sign on to the application in order to execute it.

►► For information about CA-ADS signon security, see G.3.2, “Signon security” later in this appendix.

G.2 CA-ADS compiler security

Compiler security for the Application Compiler (ADSA) and the Dialog Compiler (ADSC) prohibits unauthorized users from adding, modifying, displaying, or deleting applications and dialogs. The compilers perform a security check whenever a user begins a compiler session (that is, when a user specifies the name of a application/dialog to add, modify display, or delete). If the security check fails, the user cannot perform the specified action.

Security is established by using the Integrated Data Dictionary (IDD) when the following is true:

- IDD SECURITY is ON in the dictionary
- You are assigned the IDD authority through the AUTHORITY clause of the DDDL USER statement

►► For more information on IDD security and the DDDL USER statement, refer to the *IDD DDDL Reference*.

DDDL statements governing compiler security: Security at the compiler level restricts the actions that a user can specify for any application and dialog. Security at the compiler level is governed by the following two DDDL statements:

- **SET OPTIONS ... SECURITY FOR ADS IS ON/OFF**— Specifies whether compiler level security is in effect. If security for CA-ADS is off, the user passes the compiler level security check. If security is on and the user has not signed on to DC/UCF, the user immediately fails the security check. Otherwise, the user passes or fails the security check based on the USER statement discussed below.
- **ADD/MOD USER *user-name* ... INCLUDE/EXCLUDE AUTHORITY FOR UPDATE/ADD/MODIFY/REPLACE/DELETE/DISPLAY IS ADS**— Specifies the actions that the user has the authority to perform using the application or dialog compiler. The user passes or fails the security check depending on whether the user has specified an authorized action.

►► For more information on the SET OPTIONS and USER statements, see the *IDD DDDL Reference*.

If the user fails the application or dialog compiler level security check, the compiler displays an error message. If the user passes the security check, the compiler performs a security check at the application/dialog-specific level.

G.3 CA-ADS application security

There are two ways in which CA-ADS applications can be protected from unauthorized use.

Outside the application: Protection can be provided at the application level through the CA-IDMS central security system. For example, a user's authority to execute programs and dictionary load modules can be controlled through the CA-IDMS central security facility.

Within the application: In addition to defining security outside the application, the CA-ADS application compiler provides two security features that allow you to define security within the application:

- Security for responses
- Signon security

Security for responses is based on application activity security controlled through the CA-IDMS central security facility. Within the security facility, application activities can be defined as secured resources and authority to execute those activities granted to one or more users.

Response security, and signon security are discussed separately below.

G.3.1 Response security

Response security enables you to define security for individual application functions. To implement response security, you enter a number in the **Security class** field of the ADSA Response Definition screen. When the application is compiled, the application load module includes the activity number of each response.

At runtime, response security is enforced if the security administrator has secured activities and has defined activities that correspond to the application functions for which response security is defined. When the application issues a security check on a response, CA-IDMS central security looks for an activity definition in which the application name matches the CA-ADS application name and the activity number matches the CA-ADS response security class.

CA-ADS makes no calls to CA-IDMS central security for security class 0, which is defined always as unsecured.

►► For more information on defining and controlling application activities, refer to the *CA-IDMS Security Administration*.

If a user without execute authority for the corresponding activity, attempts to execute a secured response, the runtime system redisplay the screen from which the response was selected, along with the following message:

UNACCEPTABLE RESPONSE. PLEASE TRY AGAIN

Because the response is secured, the function invoked by the response cannot be accessed unless the security administrator has authorized the appropriate users to execute the corresponding application activity defined to CA-IDMS central security.

Response security is complemented by the CA-ADS security-tailored menus feature. At runtime, security-tailored menus list only those responses that the user has authority to select. Menus are security-tailored by selecting option 2, **Security tailored**, on the second page of the ADSA General Options screen.

G.3.2 Signon security

Signon security can be implemented for any application defined using the application compiler. With signon security, a user begins executing an application by entering a user ID and password, which the runtime system validates. To implement signon security for an application, follow the steps listed below:

1. **Specify SIGNON IS OPTIONAL or SIGNON IS REQUIRED on the second page of the ADSA General Options screen.** If signon is optional, the user can sign on before executing the application, but is not required to. If signon is required, the user must enter a valid user ID and password before executing the application.
2. **Specify the name of the signon menu function on the second page of the ADSA General Options screen.** The signon menu function is executed first when the user begins executing the application. The function displays a signon menu screen, which provides fields in which to enter a user ID and password.

►► An example of a signon menu screen is shown in 4.2.3, “System-defined menu maps.”
3. **Define an immediate response that invokes the SIGNON system function on the Response Definition screen.** When invoked at runtime, the SIGNON function validates the user ID and password entered by the user, then returns control to the signon menu function.
4. **Define the signon menu function on the Function Definition screen and any appropriate secondary screen, as follows:**
 - On the Function Definition screen, define the function as a menu function and specify the function name supplied on the Security screen. Optionally, specify that the response that invokes the SIGNON system function is the default response for the signon menu function; if this response is the default, the user need only press the [Enter] from the function at runtime to invoke the SIGNON function.
 - On the Menu Function Definition screen, specify that the menu function is a signon menu function by entering a slash (/) in the **Use signon menu** field.
 - On the Valid Responses screen, specify that the response that invokes the SIGNON system function is a valid response for the signon menu function.

Runtime processing: At runtime, processing is performed as follows:

1. When the application begins execution, the runtime system displays the signon menu function. If signon is optional, all valid responses for the function are displayed. If signon is required and menus are security tailored, only authorized responses are displayed.
2. On the signon menu screen, the user signs on by entering a user ID and password in the appropriate fields, then selecting the response that invokes the SIGNON system function. If signon is optional, the user can instead begin executing the application immediately.
3. The SIGNON system function validates the signon, then redisplay the signon menu screen with one of the following messages:
SIGNON ACCEPTED

SIGNON FAILED; UNKNOWN USER ID

SIGNON FAILED; INVALID PASSWORD
4. If the signon is accepted, all valid responses for the signon menu function are displayed; the user can execute the application. If the signon fails, the user can attempt to sign on again.

The signon menu function may be different than the function invoked by the initiating application task code. In such cases, the application begins by executing the signon menu function. The function associated with the application task code is executed when the user presses [Enter] from the redisplayed signon menu screen after signing on successfully. If signon is optional, the user can press [Enter] without signing on.

The SIGNOFF system function can be used in conjunction with signon security. When selected at runtime, the SIGNOFF function signs the user off the application, then redisplay the screen from which the function was selected. If signon is required, the next user must sign on successfully before executing the application.

The SIGNOFF function: To implement the SIGNOFF system function, perform the following steps using the application compiler:

1. Define a response that invokes the SIGNOFF system function on the Response Definition screen.
2. Make the response a valid response for the signon menu function on the Valid Responses screen.
3. Define the application structure so that the user, at runtime, can return to the signon menu function to sign off.

At runtime, when the SIGNOFF system function is invoked, the runtime system signs the user off the application, then redisplay the screen with the following message:

SIGNOFF ACCEPTED

If signon is required, the runtime system additionally blanks out all responses listed on the screen.

Appendix H. Debugging a CA-ADS Dialog

H.1	Creating a symbol table	H-4
H.2	Trace facility	H-5
H.3	Online debugger	H-7

About this appendix: To debug dialogs, you can use the CA-ADS trace facility and the CA-IDMS online debugger. This appendix explains the use of both facilities.

H.1 Creating a symbol table

Prerequisite for debugging: To use either the trace facility or the online debugger to debug a dialog, you must first compile the dialog with a symbol table. A symbol table contains information such as data field names and process command line numbers that enable the trace facility and the online debugger to execute.

How to create a symbol table: When defining the dialog using the dialog compiler, invoke the Options and Directives screen and enter a nonblank character next to the **Symbol table is enabled** prompt, as shown below:

Options and Directives	
Dialog	Version
JPKTD10	1

Type and select each option and directive. Then Enter.

Message prefix	DC
Autostatus record	ADSO-STAT-DEF-REC
Version	1
Options and directives	<div><div>Mainline dialog</div><div><input checked="" type="checkbox"/> Symbol table is enabled</div><div><input type="checkbox"/> Diagnostic table is enabled</div><div><input type="checkbox"/> Entry point is premap</div><div><input type="checkbox"/> COBOL moves are enabled</div><div><input type="checkbox"/> Activity logging</div><div><input type="checkbox"/> Retrieval locks are kept</div><div><input type="checkbox"/> Autostatus is enabled</div></div>

Enter F1=Help F3=Exit F4=Prev F5=Next

►► For more information about the Options and Directives screen, see Chapter 3, “CA-ADS Dialog Compiler (ADSC).”

In ADSOBCOM, use the symbol table option of the DIALOG expression.

►► For more information about ADSOBCOM, see Appendix D, “Application and Dialog Utilities.”

Compile the dialog. When the dialog successfully compiles, a load module with interpretable CMEs is created.

H.2 Trace facility

The CA-ADS trace facility is a debugging aid used to trace the flow of control and commands executed in a CA-ADS application at runtime.

The CA-ADS trace facility writes trace records to the DC/UCF system log as DEBUG records. Trace records can be viewed by using online PLOG or the batch print-log utility (PRINT LOG). Use the MESSAGES parameter to print the CA-ADS trace records.

Note: The PRINT LOG TRACES parameter will not print the CA-ADS trace records.

►► For more information about online PLOG, refer to *CA-IDMS System Tasks and Operator Commands*. For information about the batch print-log utility, refer to *CA-IDMS Utilities*.

Information in a trace facility report: The table below describes the information contained in a trace facility report.

Field	Contents
Dialog name	The name of the currently executing dialog.
Process name	The name of the currently executing premap or response process. ***** DIALOG-ENTRY ***** in this field documents the beginning of a dialog.
Subroutine name	The name of the process subroutine currently executing. **MAIN** in this field documents the beginning of a dialog, process, or process command that is not in a subroutine.
Sequence number	The IDD sequence number of the command currently executing. 00000000 appears in this field at the beginning of a dialog, process, or subroutine.
Process command	The process command currently executing. ENTRY in this field documents the beginning of a dialog, process, or subroutine.
Command offset	The hexadecimal offset of the command from the beginning of the dialog's fixed dialog block (FDB).
Included module name	The name of each included module.

Format of a trace facility record for a non-SQL DML statement.: The format of a trace facility record for a non-SQL DML statement is shown below.

Blank	Dialog Name	Process Name	Sub- routine Name	Sequence Number	Process Command	Command Offset	Included Module Name
0	8	17	50	61	70	79	86
							118

Format of a trace facility record for an SQL DML statement.: The trace facility report for an SQL statement follows the format below:

1. **SQL CMT** = followed by the SQL command (for example, SELECT).
2. **CODE** = followed by the appropriate SQL code
3. **ERROR** = followed by the 5-digit error.

Below this information is the database message passed from the SQLCA.

►► For information about the SQLCA, see *CA-IDMS SQL Programming*.

Initiating the trace facility: You can initiate the trace facility in one of these ways:

- Use the TRACE keyword in the runtime system initiating statement when requesting execution of the application
 - For more information about the TRACE keyword, see 4.1, “Initiating the CA-ADS runtime system.”
- Coding the TRACE command in the dialog process
 - For documentation of the TRACE command, see 20.6, “TRACE.”

Specifying TRACE when initiating CA-ADS results in tracing the execution of all dialogs in the application that have been compiled with a symbol table. You use TRACE and TRACE OFF in process logic to limit the trace.

The system log: Using the CA-ADS trace facility can fill the DC/UCF system log quickly. Information on the entire application is collected for each process command in dialogs that have a symbol table enabled and TRACE=ALL specified. A record is written to the system log for each command.

To avoid overloading the system log, the system log can be defined to sequential log files instead of the DDLDCLOG area. Assigning the system log to sequential log files facilitates offloading the system log when it becomes full.

►► More information about log files can be found in *CA-IDMS System Generation*.

H.3 Online debugger

What you can do: The online debugger enables the application developer to interrupt execution of a dialog's premap or response process, display and change the contents of data fields, and restart execution from any point in the interrupted dialog. If a dialog aborts during a debugger session, the application developer can review the contents of the data fields at the time of the abend, change the contents of the data fields, and resume dialog execution at any point. For example, a breakpoint can be set at line 200 in a premap process and can specify that an interruption is to occur every second time the process command is executed at runtime.

If CA-ADS encounters a potential breakpoint at runtime, it passes control to the online debugger. If the conditions for interrupting the dialog are not met, control returns to the runtime system and execution continues. If the conditions are met, the online debugger keeps control and allows the application developer to perform functions such as reviewing and modifying data fields, modifying breakpoint specifications, aborting dialog execution, and resuming execution at a specified point.

If a dialog aborts during a debugger session, the CA-ADS runtime system displays a special version of the Dialog Abort Information screen and then links to the online debugger.

►► For more information, see 4.6, "Dialog Abort Information screen."

The special screen version allows the application developer to continue the debugging session for the dialog. The application developer can enter debugger commands at the prompt that appears on the screen.

Procedures: Procedures for debugging a CA-ADS dialog are the same as those used with any other program running under DC/UCF: once the dialog is defined to the debugger with the DEBUG command, debugging procedures can take place. It is easier, however, to find dialog records or to find the command elements (CMEs) for breakpoints when the load module is generated in interpretable code and symbol recognition is in effect.

►► For more information about debugging, refer to *CA-IDMS Online Debugger*.

Recommended steps: It is recommended that you take the following steps when debugging a CA-ADS dialog:

1. Create a symbol table for the dialog
2. Compile the dialog
3. Run ADSORPTS for the dialog
4. Issue the DEBUG task code to invoke the debugger
5. Define the dialog to the debugger

6. Set breakpoints (as required)
7. Issue the EXIT command to leave the debugger
8. Invoke the CA-ADS runtime system and execute the dialog
9. Continue processing the dialog
10. Issue the EXIT or QUIT command when debugging is completed

Run ADSORPTS for the dialog: Run the CA-ADS Dialog Report (ADSORPTS) for the given dialog. Specify the PROCESS and/or FDBLIST options when submitting the report: PROCESS displays the sequence line numbers that are assigned to the process source; FDBLIST provides the line numbers (SEQ#) and the offsets of the CMEs. The address or line number of a CME can then be used to set a valid breakpoint within the premap or response process.

►► For more information on the ADSORPTS utility, see Appendix B, “CA-ADS Dialog and Application Reporter.”

Issue the DEBUG task code: Issue the DEBUG task code to invoke the debugger.

►► For information on initiating a debugger session, refer to *CA-IDMS Online Debugger*.

Define the dialog to the debugger: Define the dialog to the debugger by issuing a command similar to the one in the following example:

```
DEBUG>  
debug dialog medduins
```

The debugger responds to the above command with the following message:

```
DEBUG DIALOG MEDDUINS  
DEBUG> DEBUGGING INITIATED FOR MEDDUINS VERSION 1  
DEBUG>
```

If you omit the word *dialog*, the debugger issues an error message:

```
DEBUG>  
debug medduins  
DEBUG MEDDUINS  
DEBUG> INCONSISTENT ENTITY TYPE  
          - MEDDUINS VERSION 1 DEFINED AS A DIALOG  
DEBUG>
```

This message indicates that the debugger has tried to process MEDDUINS as a program but can only find a PDE (program descriptor element) that defines MEDDUINS as a dialog. The command needs to be modified to state that MEDDUINS is a dialog:

```
DEBUG>
debug dialog medduins
```

Set breakpoints: Set breakpoints as required. Breakpoints must be set at line numbers or addresses that contain valid command instructions (valid CMEs).

►► For more information about setting breakpoints, refer to *CA-IDMS Online Debugger*.

Issue EXIT: Issue the EXIT command and leave the debugger.

Invoke the CA-ADS runtime system: Invoke the CA-ADS runtime system and execute the dialog in the standard manner.

When a breakpoint is encountered during the execution of the dialog, a message appears on the screen that identifies the breakpoint. The DEBUG> prompt or menu mode screen is displayed, signalling that you are now in the runtime phase of the debugger and can enter any of the debugger commands except DEBUG.

►► For complete information on command syntax, refer to *CA-IDMS Online Debugger*.

Continue processing the dialog: Continue processing the dialog. When the single command RESUME is issued without any qualifying parameters, processing continues from the current CME (the instruction immediately following the breakpoint). When a RESUME *debug-expression* is issued, processing resumes at the address specified by the expression.

When you issue a RESUME *dialog-expression* command from a point within the main body of the dialog process, the debug expression must resolve to an address also within the main body of the dialog. Similarly, when a RESUME *dialog-expression* is issued from a subroutine, the debug expression must resolve to an address within the same subroutine. Results are unpredictable when execution is not resumed in accordance with these rules.

In the event of an abend: In the event of an abend, you see the CA-ADS Debug screen with dialog abort information, the DEBUG> prompt, and the menu mode selection area. Any valid debugger command can be entered on the prompt line or can be selected from the menu. When a selection is made from the menu, the debugger automatically operates in menu mode and displays the specified screen. A sample Debug screen is shown below. Note the DEBUG> prompt and menu selection area located at the bottom of the screen. All commands, except DEBUG, can be issued in response to the prompt or can be selected in the menu area:

```

      CA-ADS RELEASE 15.0          *** DIALOG ABORT INFORMATION ***   DBUG
DC175020 APPLICATION ABORTED. PGM CHECK (DATA EXCEPTION).

DATE.....: 91.220      TIME.....: 17:10:23.55      TERMINAL.....: LV81001

ERROR OCCURRED IN DIALOG.....: MISINCD
      AT OFFSET.....: 3D8
      IN PROCESS.....: MIS-MAIN1
      AT IDD SEQ NO. : 000000100      INTERNAL COMMAND: 2
      INCLUDED MODULE : MIS-INC1      VERSION: 1
      VERSION: 1

SEQUENCE
NUMBER:      SOURCE :
00000000
00000100      ADD 1 TO MIS-NUM.
00000200      ! THIS IS MIS-INC1

DEBUG>
NEXT _ ACTIVITY OR _ HELP:
      _ AT      _ LIST      _ SET      _ SNAP      _ RESUME      _ DEBUG      _ WHERE

      _ EXIT      _ PROMPT      _ QUIT      _ IOUSER
HELP SCREENS: _ USAGE      _ SYMBOLS      _ KEYS

```

Issue the **RESUME ABEND** or a **RESUME *debug-expression*** command to continue processing the dialog.

QUIT or EXIT: Issue the **QUIT** or **EXIT** command when debugging is completed. **QUIT** clears the debugger control blocks and ends the debugger session; **EXIT** returns you to the **ENTER NEXT TASK CODE** prompt, but leaves the control blocks intact so that the debugger session can continue.

Index

Special Characters

\$BACKWARD condition 8-23
\$BATCH condition 8-16
\$DETAIL condition 8-23
\$DETAIL-NOT-FOUND condition 8-23
\$END-OF-DATA condition 8-23
\$END-OF-FILE condition 8-6
\$ERROR-COUNT field 11-9
\$FORWARD condition 8-22
\$HEADER condition 8-23
\$INPUT-COUNT field 11-9
\$IOERROR condition 8-6
\$MAXIMUM-DETAILS-PUT condition 8-24, 17-32
\$MESSAGE field 8-18, 17-33
\$ONLINE condition 8-16
\$OUTPUT-COUNT field 11-9
\$PAGE field 8-18, 17-24
\$PAGE-READY condition 8-22
\$RESPONSE field 8-18
#EFMBIFS macro F-28
#EFUNMOD macro F-31
#EFUNMST macro F-27

A

ABORT command 20-4
ABSOLUTE-VALUE 7-11
ACCEPT command 16-12, 20-8
access module 3-21
activity logging E-3—E-8
 activity logging records E-3—E-6
 function commands E-6
 function numbers E-6
ADB
 See application definition block (ADB)
ADD command 13-6
ADSA
 See application compiler (ADSA)
ADSC
 See dialog compiler (ADSC)
ADSL 1-30
ADSM 1-31
ADSO-APPLICATION-GLOBAL-RECORD A-4—A-14
 AGR-CURRENT-RESPONSE 4-19, 15-17
 definition of A-4
 usage A-4
ADSO-APPLICATION-MENU-RECORD
 at runtime 4-8
 definition of A-15
 usage A-15
ADSO-STAT-DEF-REC record 10-9
ADSOBCOM D-4—D-36
 control statements D-5—D-29
 JCL and command statements D-30
ADSOBSYS D-37—D-48
 ADSOOPTI load module D-37
 JCL and command statements D-39
ADSOBTAT D-48—D-56
 JCL and command statements D-51
 task application table (TAT) D-48
ADSOMSON menu map 4-9
ADSOMUR1 menu map 4-9
ADSOMUR2 menu map 4-9
ADSOOPTI load module
 See ADSOBSYS
ADSORPTS
 application reports B-15
 control statements B-16—B-23
 dialog debugging B-13
 dialog reports B-4—B-5
ADSOTATU D-57—D-59
AFACT-057 record E-4—E-6
AGR-CURRENT-RESPONSE
 See ADSO-APPLICATION-GLOBAL-RECORD
ALLOCATE command 21-10
ALLOWING clause 10-7
AMR-RESPONSE-FIELD
 See ADSO-APPLICATION-MENU-RECORD
APPC (Advanced Program to Program
 Communication) 21-3
APPC status codes 21-30
APPCCODE status code 21-30, 21-31
APPCERC status code 21-30, 21-31
application compiler (ADSA) 2-19, 2-27
 control key assignments 2-10
 Function Definition (Menu) screen 2-32—2-37
 Function Definition (Program) 2-30
 Function Definition (Program) screen 2-32
 Function Definition screen 2-27
 General Options 2-14—2-19
 General Options screen—Page 2 2-16—2-19
 General Options—Page 1 2-14—2-16
 Global Records screen 2-37
 Response Definition screen 2-23

- application compiler (ADSA) (*continued*)
 - Response/Function List screen 2-19—2-23
 - secondary screens 2-32
 - Task Codes screen 2-39
- application compiler sequence
 - screen sequence 2-7
- application compiler session
 - invoking 2-4
 - screen sequence 2-10
 - suspending 2-10
- application definition block (ADB) 20-12
- application reporter
 - See* ADSORPTS
- application response
 - See* response
- application security G-5—G-7
- application structure
 - levels of 15-6
 - mainline dialogs 15-7
- application thread
 - definition of 15-5
 - menu stack 15-7
 - nonoperative dialogs 15-6
 - operative dialogs 15-6
- applications
 - compiling of 2-10
 - defining global records 2-37
 - defining responses and functions 2-19
 - defining task codes 2-39
 - specifying control blocks 2-30
 - specifying menus 2-32
 - specifying record buffers 2-30
- APPLICATIONS statement B-16—B-18
- arc cosine values 7-12
- arc sine values 7-13
- arc tangent values 7-14
- AREPORTs B-3
- arithmetic built-in functions
 - ABSOLUTE-VALUE 7-11
 - INVERT-SIGN 7-30
 - LOG-BASE-10 7-34
 - LOG-BASE-E 7-34
 - MODULO 7-35
 - next integer equal or higher 7-36
 - next integer equal or lower 7-37
 - NUMERIC 7-38
 - RANDOM-NUMBER 7-40
 - sign inversion 7-30
 - SIGN-VALUE 7-45
 - SQUARE-ROOT 7-47

- arithmetic commands
 - See also* arithmetic expression
 - ADD 13-6
 - COMPUTE 13-7
 - DIVIDE 13-8
 - MULTIPLY 13-10
 - SUBTRACT 13-11
 - summary of 13-3
- arithmetic expressions
 - binary operations 6-3
 - coding rules 6-6
 - operands 6-3
 - order of evaluation 6-5
 - unary operations 6-3
 - variable data fields 6-5
 - WHERE clause 16-66
- Assembler
 - See* user program
- assigned key 2-21
- assignment command
 - MOVE 13-12
- automatic editing 4-22, 8-20
- autostatus facility 10-4—10-5
 - status codes 10-4

B

- BACKWARD function 1-9
 - See also* system functions
- batch control event conditions
 - \$END-OF-FILE (\$EOF) 8-6
 - \$IOERROR (\$IOERR) 8-6
- batch dialog compiler
 - See* ADSOBCOM
- batch processing
 - \$ERROR-COUNT 11-9
 - \$INPUT-COUNT 11-9
 - \$OUTPUT-COUNT 11-9
- binary data 5-11
- BIND PROCEDURE command 16-19
- BS2000/OSD JCL
 - ADSOBCOM D-35
 - ADSOBSYS D-46
 - ADSOBTAT D-55
 - ADSORPTS B-29
- built-in functions 4-34—4-39
 - See also* functions
 - calling user-defined functions F-42
 - changing invocation names F-3
 - coding user-defined functions F-42
 - creating user-defined functions F-41

built-in functions (*continued*)

- data type conversion 7-4
- date formats 7-6
- error processing 7-4
- internal structure F-4
- invocation name 7-3, F-40
- omitted optional parameter 7-4
- parameters 7-4
- runtime processing F-24
- user-defined 7-5, F-41
- with LRF F-42

C

CA-ADS

- conversion rules 13-12

CA-ADS comment character 5-9

CA-ADS dialog and application reporter

- See* ADSORPTS

CA-ADS statistics block

- See* dialog statistics

CA-IDMS statistics block 16-18

CA-OLQ 4-33

CALL command 19-4

CHANGED condition 8-19

characteristic 9-10

checkouts

- explicit 1-28
- implicit 1-29
- listing 1-30
- modifying with ADSM 1-31
- releasing with ADSM 1-31

checkpoint 4-26, 16-59

- database 16-59

- queue 16-59

- scratch 16-59

CLOSE command 17-10

COBOL

- See also* user program

- conversion rules 13-12

COBOL moves 3-16, 5-17, 13-6, 13-7, 13-8, 13-9,
13-10, 13-12, 13-13, 13-14

coding

- arithmetic expressions 6-6
- CA-ADS comment character 5-9
- general rules 5-8
- SQL comment character 5-9

command status condition

- ERROR-STATUS 8-7

command-statements

- DO 14-5

command-statements (*continued*)

- ELSE 14-5

- END 14-5, 14-10

- REPEAT 14-10

- THEN 14-5

commands 12-4, 14-3

- See also* Logical Record Facility database access

- See also* navigational database access

- See also* process commands

- See also* VM/ESA commands

- ABORT 20-4

- ACCEPT 16-12, 20-8

- ADD 13-6

- arithmetic 13-3

- assignment 13-12

- BIND PROCEDURE 16-19

- CALL 19-4

- CLOSE 17-10

- COMMIT 16-20

- COMPUTE 13-7

- conditional 14-3

- CONNECT 16-22

- CONTINUE 15-10

- control 15-3—15-38

- DEFINE 19-5

- DELETE QUEUE 18-7

- DELETE SCRATCH 18-17

- DISCONNECT 16-25

- DISPLAY 15-12

- DIVIDE 13-8

- ERASE 16-27, 16-68

- EXECUTE NEXT FUNCTION 15-17

- EXIT 14-4

- FIND/OBTAIN 16-30

- GET 16-44

- GET DETAIL 17-28

- GET QUEUE 18-9

- GET SCRATCH 18-19

- GOBACK 19-6

- IF 14-4

- INCLUDE 12-8

- INITIALIZE RECORDS

- INVOKE 15-19

- KEEP 16-46

- KEEP LONGTERM 16-47

- LEAVE 15-22

- LINK 15-24

- LRF 16-64—16-76

- map modification 17-3

- MODIFY 16-53, 16-69

- MULTIPLY 13-10

commands (*continued*)

- navigational database access 16-5—16-63
- NEXT 14-8
- OBTAIN 16-70
- pageable map 17-3
- PUT DETAIL 17-30
- PUT QUEUE 18-12
- PUT SCRATCH 18-22
- queue management 18-3
- READ TRANSACTION 15-30
- RETRUN 15-31
- RETURN DB-KEY 16-57
- ROLLBACK 16-59
- scratch management 18-3
- SNAP 20-11
- STORE 16-60, 16-75
- subroutine control 19-3
- SUBTRACT 13-11
- summary of 12-4
- TRACE 20-13
- TRANSFER 15-34
- utility 20-3
- WHILE 14-10
- WRITE PRINTER 20-14
- WRITE TO LOG/OPERATOR 20-18
- WRITE TRANSACTION 15-36

COMMIT command 16-20

comparison conditions

- CONTAINS 8-10
- mask characters 8-11
- MATCHES 8-10
- operators 8-10

compiler (ADSA)

- security G-4

compiler (ADSC)

- security G-4

COMPUTE command 13-7

CONCATENATE built-in function 7-15

conditional commands

- EXIT 14-4
- IF 14-4
- NEXT 14-8
- WHILE 14-10

conditional expressions

- \$MESSAGE field 8-18
- \$PAGE field 8-18
- \$RESPONSE field 8-18
- batch control event condition 8-6
- command status condition 8-7
- comparison condition 8-10
- cursor position condition 8-12

conditional expressions (*continued*)

- dialog execution status condition 8-14
- environment status condition 8-16
- for maps 17-4
- level-88 condition 8-17
- map field status condition 8-18
- map paging status conditions 8-22
- operators in 8-4
- order of precedence 8-4
- set status condition 8-25
- summary 8-5

CONFIRM command 21-13

CONFIRMED command 21-14

CONNECT command 16-22

constants 9-3

- figurative constants 9-4
- fixed-point numeric literals 9-9
- floating-point numeric literals 9-9
- graphic literals 9-6
- multibit binary 9-7
- nonnumeric literals 9-8
- numeric literals 9-9

CONTAINS condition 8-10

CONTINUE command 15-10

control commands

- BIND PROCEDURE 16-19
- CLOSE 17-10
- CONTINUE 15-10
- DISPLAY 15-12
- EXECUTE NEXT FUNCTION 15-17
- INVOKE 15-19
- LEAVE 15-22
- LEAVE APPLICATION 15-22
- LINK 15-24
- READ TRANSACTION 15-30
- RETURN 15-31
- TRANSFER 15-34
- WRITE TO LOG/OPERATOR 20-18
- WRITE TRANSACTION 15-36

control event 2-21, 8-6

- See also* assigned key

control keys

- See also* application compiler
- See also* dialog compiler (ADSA)
- default assignments 4-21

CONTROL SESSION command 21-15

control statements

- See also* ADSORPTS
- ADSOBSYS D-37
- ADSOBTAT D-49
- COMPILE D-6

control statements (*continued*)
 DECOMPILE D-8
conversation, ending 21-25
conversion
 CA-ADS rules 13-12
 COBOL rules 13-12
cosine values 7-16
currency
 See also NOSAVE
 database 15-7, 16-5
 index 16-58
 of area 16-5
 of record type 16-5
 of run unit 16-5, 16-33
 of set type 16-5
 queue 18-5
 scratch requests 18-15
cursor position condition
 CURSOR-COLUMN 8-12
 CURSOR-ROW 8-12
cursor position data field
 CURSOR-COLUMN 11-8
 CURSOR-ROW 11-8
CURSOR-COLUMN condition 8-12
CURSOR-ROW condition 8-12

D

data dictionary
 AFACT-057 E-4—E-6
 LR-190 E-4
 LRACT-193 E-4—E-6, E-8
 organization E-4
 RCDACT-059 E-4—E-6, E-8
 SETACT-061 E-4—E-6
 SSA-024 E-4
 SSOR-034 E-4
 SSR-032 E-4
Data Dictionary Reporter
 See AREPORTs
data types
 available to CA-ADS 5-10
 binary 5-11
 conversion of 5-16
 conversion rules 13-12
 definition of 5-10
 doubleword binary 5-11
 EBCDIC 5-11
 examples of 5-14
 floating point 5-12
 fullword binary 5-11

data types (*continued*)
 group 5-12
 halfword binary 5-11
 multibit binary 5-13
 packed decimal 5-13
 zoned decimal 5-13
database
 access of 4-25
 CA-IDMS statistics block 16-18
 CALC key 16-31
 checkpoint 16-20
 currency 16-5
 db-key 16-12, 16-14, 16-34, 16-58
 location modes 16-62
 modification of CALC elements 16-54
 modification of sort-control elements 16-54
 monitoring activity 16-47—16-51
 set membership options 16-22
 statistics 16-17
 usage modes 16-55
database access 4-24
 Logical Record Facility (LRF) 16-64
database activity
 See activity logging
database commands
 See also Logical Record Facility database access
 See also navigational database access
 implicit E-3
database currency
 See currency
 See NOSAVE
date built-in functions
 DATECHG 7-17
 DATEDIF 7-20
 DATEOFF 7-21
 GOODDATE 7-25
 TODAY 7-54
 TOMORROW 7-56
 WEEKDAY 7-62
 YESTERDAY 7-66
date formats
 calendar 7-6
 European 7-6
 Gregorian 7-6
 Julian 7-6, 11-8
date offset 7-21
DATECHG built-in function 7-17
DBCS data
 as a graphic literal 9-6
 as a nonnumeric literal 9-8
 storage of 13-4

deadlock 4-26
 DEALLOCATE command 21-17
 debugging 4-32
 See also ADSORPTS
 See also online debugger
 See also trace facility
 DEFINE command 19-5
 DELETE QUEUE command 18-7
 DELETE SCRATCH command 18-17
 Design guidelines 21-25
 detail area 17-21
 diagnostic screen
 See Dialog Abort Information screen
 diagnostic table 3-15
 dialog
 See also ADSOBCOM
 FDB B-5—B-11
 Dialog Abort Information screen 4-28, B-14, H-7
 enabling of 20-6
 dialog compiler
 See ADSOBCOM
 dialog compiler (ADSC)
 control keys 3-7
 Dialog Summary Report screen 1-26
 Dialog Summary screen 1-26
 Map Image screen 1-26
 Options and Directives 3-13
 screens 3-10
 session 3-4
 dialog compiler session
 invoking 3-4
 suspending 3-8
 dialog execution status
 FIRST-TIME 8-14
 dialog expression (ADSOBCOM) D-10—D-29
 dialog function 1-9
 See also function
 dialog reporter
 See ADSORPTS
 Dialog Selection screen 4-4
 dialog statistics
 CA-ADS statistics block C-5
 checkpoint interval C-10
 enabling of C-8
 runtime collection and writing C-11
 selecting C-9
 statistics block identifiers C-11
 statistics reporting C-12
 transaction statistics block C-4
 Dialog Summary screen 1-26
 Map Image screen 1-26

dialog, location of allocated 21-25
 dialogs
 mainline 4-3
 DIALOGS statement B-19—B-21
 DISCONNECT command 16-25
 DISPLAY command 15-12
 mapout rules 15-14
 status test outcome 8-14
 DIVIDE command 13-8
 DO command-statement 14-5
 double-byte character set
 See DBCS
 doubleword binary data type 5-11
 dumps
 snap dumps 20-6

E

EBCDIC data type 5-11, 13-4
 EDIT IS ERROR/CORRECT condition 8-18
 ELSE command-statement 14-5
 END command-statement 14-5, 14-10
 environment status condition
 \$BATCH 8-16
 \$ONLINE 8-16
 ERASE command 16-27, 16-68
 ERASED condition 8-20
 error expressions 10-6
 error handling
 ADSO-STAT-DEF-REC 10-9
 ALLOWING clause 10-7
 autostatus 10-4—10-5
 built-in functions F-25
 error expressions 10-6
 level-88 condition names 10-9
 site defined status definition record 10-10
 STATUS clause 10-9
 status definition record 10-9
 system-defined status definition record 10-9
 error messages
 suppression of 17-17
 ERROR-STATUS condition 8-7
 exclusive usage mode 16-55
 EXECUTE NEXT FUNCTION command 8-14
 mapless dialog 15-17
 EXECUTE ON EDIT ERRORS command 8-20
 execution modes 2-16, 3-3
 EXIT command 14-4
 explicit checkouts 1-28
 explicit releases 1-28

expression description element
 See XDE module
extended run units 4-25—4-27
 See also run unit
 checkpoint 4-26
 deadlock 4-26
EXTRACT built-in function 7-23

F

fast mode 2-16
FDB
 See fixed dialog block
 See fixed dialog block (FDB)
figurative constants 9-4
FIND/OBTAIN
 FIND 16-31
FIND/OBTAIN command
FIRST-TIME condition 8-14, 15-34
FIX built-in function 7-24
fixed dialog block (FDB) 20-11
 contents of B-5—B-11
fixed-point numeric literals 9-9
floating point data types
 display 5-12
 internal long 5-12
 internal short 5-12
floating-point numeric literals 9-9
flow of control 4-19—4-22
 automatic editing 4-22
 default control key assignments 4-21
footer area 17-21
FORWARD function 1-9
 See also system functions
fullword binary data type 5-11
function
 See dialog function
 See menu function
 See menu/dialog function
 See program function
Function Definition (Dialog) screen 2-27
Function Definition (Menu) screen 2-32
Function Definition (Program) screen 2-30
functions
 See built-in functions

G

G-literals
 See graphic literals

General Options screen—Page 2 2-16—2-19
General Options—Page 1 2-14—2-16
GET command 16-44
GET DETAIL command 8-23, 17-28
GET QUEUE command 18-9
GET SCRATCH command 18-19
Global Records screen (ADSA) 2-37
GOBACK command 19-6
GOODDATE built-in function 7-25
GOODTRAILING built-in function 7-26
graphic literals 9-6
group data type 5-12

H

halfword binary data type 5-11
header area 17-21
HELP function 1-9
 See also system functions
help screen 4-16

I

ICTL statement
 ADSOBCOM D-4
 ADSOBSYS D-37
IDENTICAL condition 8-19
IF command 14-4
implicit checkouts 1-29
implicit releases 1-29
IN ERROR condition 8-18, 8-20
INCLUDE command 12-8
INDEX built-in function 7-48
INITCAP built-in function 7-27
INITIALIZE RECORDS command 20-10
INSERT built-in function 7-28
INSERT directive 12-8
intermediate result area (IRA) F-24
INVERT-SIGN built-in function 7-30
INVOKE command 15-19
 status test outcome 8-14
ISEQ statement
 ADSOBCOM D-4
 ADSOBSYS D-37

J

JCL
 See BS2000/OSD JCL
 See OS/390 JCL
 See VM/ESA commands
 See VSE/ESA JCL

K

KEEP commands 16-46
KEEP LONGTERM command

L

LEAVE APPLICATION command 15-22
LEAVE command 15-22
 status test outcome 8-14
LEFT-JUSTIFY built-in function 7-31
length
 See STRING-LENGTH built-in function
level-88 condition 8-7, 8-17
LIKE built-in function 7-32
LINK command
 linking to OLQ 4-33
 nesting 15-26
 status test outcome 8-14
 with a user program 15-26
LIST statement B-21
literal
 See constants
local mode processing
 SYSIDMS parameter file B-24
location mode
 CALC 16-62
 DIRECT 16-62
 VIA 16-62
log
 See activity logging
LOG-BASE-10 built-in function 7-34
LOG-BASE-E built-in function 7-34
logarithms
 See arithmetic built-in functions
Logical Record Facility
 linking to dialog with LRF subschema 15-27
logical records
 See also Logical Record Facility database access
 in database access 16-64
 path 16-64
LR-190 record E-4
LRACT-193 record E-4—E-8
LRF
 path status 16-72
LRF commands
 ERASE 16-68
 MODIFY 16-69
 OBTAIN 16-70
 ON command 16-71
 STORE 16-75

LRF commands (*continued*)

 WHERE clause 16-65

LU 6.2 21-3

M

mainline dialogs 15-7
mantissa values 9-10
map field status conditions
 ALL BUT 8-19
 CHANGED 8-19
 ERASED 8-20
 EXCEPT 8-19
 IDENTICAL 8-19
 IN ERROR 8-18, 8-20
 pageable map considerations 8-20
 TRUNCATED 8-20
Map Image screen 1-26
map modification commands
 ATTRIBUTES 17-5—17-9
 MODIFY MAP 17-12—17-20
map paging session 17-23
map paging status condition 8-22
 See also pageable maps
maps
 See also Map Image screen
 See also map modification commands
 See also menu map
 See also pageable maps
 See also screens
 conditional expressions 17-4
 output data options 17-16
 permanent modifications 17-7, 17-13
 suppressing error message 17-17
 temporary modifications 17-7, 17-13
mask character 7-32, 8-11, 16-67
master function table F-4, F-5
MATCHES condition 8-10
matching string 7-32
menu definition 4-8
menu function 1-9
 See also function
menu maps
 ADSOMSON 4-9
 ADSOMUR1 4-9
 ADSOMUR2 4-9
 signon 4-13—4-16
 site-defined 4-9
 system-defined 4-10—4-13
menu stack 15-7

menu/dialog function 1-9
 See also function
message codes 15-15
messages 15-15
modified data tags (MDTs)
 resetting 17-14
 setting for map fields 17-19
MODIFY command 16-53, 16-69
MODIFY MAP
MODIFY MAP command 8-20, 17-12
modules
 See process modules
MODULO built-in function 7-35
MOVE command 13-12
multibit binary constants 9-7
multibit binary data type 5-13
multiple databases
 accessing 11-6
MULTIPLY command 13-10

N

native VSAM data sets 16-7—16-9
 currency requests 16-15
 set status condition 16-8
 with CONNECT 16-23
 with DISCONNECT 16-26
 with ERASE 16-27
 with FIND/OBTAIN OWNER 16-38
 with MODIFY 16-55
natural logarithm 7-34
navigational DML commands
 ACCEPT 16-12
 COMMIT 16-20
 CONNECT 16-22
 DISCONNECT 16-25
 ERASE 16-27
 FIND/OBTAIN 16-30
 GET 16-44
 KEEP 16-46
 MODIFY 16-53
 READY 16-55
 RETURN DB-KEY 16-57
 ROLLBACK 16-59
 STORE 16-60
nesting 15-26
NEXT command 14-8
NEXT-INTEGER-EQUAL-HIGHER built-in
 function 7-36
NEXT-INTEGER-EQUAL-OR-LOWER built-in
 function 7-37

nonnumeric literals 9-8
NUMERIC built-in function 7-38
numeric fields 13-4
numeric literals 9-9

O

OBTAIN command 16-70
OCB
 See online control block (OCB)
OCTL statement
 ADSOBCOM D-4
 ADSOBSYS D-37
OLQ
 See CA-OLQ
ON command 16-71
online control block (OCB) 20-11
online debugger H-7—H-10
online help
 in CA-ADS applications 4-8
 in CA-ADS compilers 1-32
online terminal block (OTB) 20-11
online terminal block extension (OTBX) 20-11
online work area (OWA) 20-11
Options and Directives 3-13
OS/390 JCL
 ADSOBCOM D-30
 ADSOBSYS D-39
 ADSOBTAT D-51
 ADSORPTS B-25
OTB
 See online terminal block (OTB)
OTBX
 See online terminal block extension (OTBX)
OWA
 See online work area (OWA)

P

packed decimal data type 5-13
pageable map commands
 GET DETAIL 17-28
 PUT DETAIL 17-30
pageable maps
 \$PAGE field 17-24
 areas of 17-21
 Auto display specification 3-19
 Backpage specification 3-19
 flow of control 17-25
 map paging dialog options 3-18, 17-27—17-28
 map paging session 17-22

pageable maps (*continued*)
 severity codes 17-33
 UPDATE specification 3-19
parameters for process commands
 keywords 5-7
 variable terms 5-7
path status (LRF) 16-72
PF keys
 See application compiler
PL/I
 See user program
POP function 1-9
 See also system functions
POPTOP function 1-9
 See also system functions
PREPARE-TO-RECEIVE command 21-19
printer output 17-15
process commands
 See also commands
 coding of 5-8
 comment character 5-9
 quoted strings 5-9
process modules
 premap 5-5
 response 5-5
processing
 cooperative 21-3
program
 See user program
program function 1-9
 See also function
 See also user program
protected usage mode 16-55
PUT DETAIL command 8-20, 17-30
PUT QUEUE command 18-12
PUT SCRATCH command 18-22

Q

queue management commands
 DELETE QUEUE 18-7
 GET QUEUE 18-9
 PUT QUEUE 18-12
queue records 18-5
QUIT function 1-9
 See also system functions
quoted strings
 See process commands

R

RANDOM-NUMBER built-in function 7-40
RBB
 See record buffer block (RBB)
RCDACT-059 record E-4—E-8
READ TRANSACTION command 15-30
READY command 16-55
RECEIVE-AND-WAIT command 21-20
record buffer block (RBB) 20-12
record locking
 queue 18-5
record locks
 deadlock conditions 16-10
 exclusive 16-9
 explicit 16-9
 implicit 16-9
 long-term explicit 16-9
 release of 16-20
 retrieval locks 16-10—16-12
 shared 16-9
records
 queue 18-5
 scratch 18-15
recovery 16-59
releases
 explicit 1-28
 implicit 1-29
releasing entities 1-29
remainder values 7-35
REPEAT command-statement 14-10
repeat string 7-50
REPLACE built-in function 7-42
replace string 7-42
reports
 See ADSORPTS
 See AREPORTs
REQUEST-TO-SEND command 21-21
REQUEST-TO-SEND-RECEIVED system field 21-30,
 21-34
reset keyboard 17-14
response 1-9
 runtime selection 4-9
Response Definition screen (ADSA) 2-23
response process
 security 3-29
Response/Function List screen 2-19
Response/Function List screen (ADSA) 2-19
Response/Function search 2-22
responses
 runtime selection 4-16

responses (*continued*)

security 4-20

RETURN command 15-31

status test outcome 8-14

RETURN DB-KEY command 16-57

RETURN function 1-9

See also system functions

RHDCEVBF source module F-28

RIGHT-JUSTIFY built-in function 7-44

ROLLBACK command 16-59

run units 4-25

extended 4-25—4-27

usage modes 4-26

runtime system

ADSOMAIN 4-3

ADSORUN1 4-3

initiation of 4-3—4-7

S

scratch area 18-15

scratch management commands

DELETE SCRATCH 18-17

GET SCRATCH 18-19

PUT SCRATCH 18-22

scratch records 18-15

screens

See also application compiler

See also dialog compiler

See also maps

application compiler 2-11

Dialog Abort Information screen 4-28—4-31

Dialog Selection screen 4-4

Dialog Summary Report screen 1-26

Dialog Summary screen 1-26

Function Definition (Dialog) screen 2-27

Function Definition (Menu) screen 2-32

Function Definition (Program) 2-30

General Options—Page 1 (ADSA) 2-14—2-16

General Options—Page 2 (ADSA) 2-16—2-19

Global Records 2-37

help 4-16—4-18

Map Image screen 1-26

Menu definition 4-8

Options and Directives (ADSC) 3-13

Response Definition 2-23

Response/Function List 2-19

Response/Function List screen 2-19

screens 3-10

task codes 2-39

TAT Update Utility D-58

SEARCH statement B-22

search, Response/Function 2-22

security

See also application security

ADAPGOP2 2-17

ADSOBCOM D-4

General Options screen—Page 2 2-16—2-19

response G-5

response process 3-29

security-tailored menus G-6

signon G-6—G-7

SEND-DATA command 21-22

SEND-ERROR command 21-24

SEND/RECEIVE commands 21-3

summary 21-9

SET condition 8-25

set membership options 16-22

SETACT-061 record E-4—E-6

sign inversion 7-30

SIGN-VALUE built-in function 7-45

SIGNOFF function 1-9, G-7

See also system functions

SIGNON function 1-9, G-6

See also system functions

SIGNON statement (ADSOBCOM) D-5

sine values 7-46

SNA (Systems Network Architecture) 21-3

SNAP command 20-11

sort key 16-42

sorted set 16-42

source code

See process modules

source module

See process modules

SQL

access module 3-21

compliance 3-22

SQL comment character 5-9

SQUARE-ROOT 7-47

SREPORTs C-12

SSA-024 area E-4

SSOR-034 set E-4

SSR-032 record E-4

statistics

See dialog statistics

status codes 21-31

status definition record

ADSO-STAT-DEF-REC 10-9

level-88 condition names 10-9

site defined 10-10

STATUS clause 10-9

status definition record (*continued*)

 system defined 10-9

step mode 2-16

storage

 management 4-40

 XA 4-40

STORE command 16-60, 16-75

string built-in functions

 CONCATENATE 7-15

 EXTRACT 7-23

 FIX 7-24

 INDEX 7-48

 INITCAP 7-27

 INSERT 7-28

 LEFT-JUSTIFY 7-31

 LIKE 7-32

 REPLACE 7-42

 RIGHT-JUSTIFY 7-44

 STRING-INDEX 7-48

 STRING-LENGTH 7-49

 STRING-REPEAT 7-50

 SUBSTRING 7-51

 TOLOWER 7-55

 TOUPPER 7-57

 TRANSLATE 7-59

 VERIFY 7-61

 WORDCAP 7-65

string verification 7-61

STRING-LENGTH built-in function 7-49

STRING-REPEAT built-in function 7-50

subroutine control commands

 CALL 19-4

 DEFINE 19-5

 GOBACK 19-6

Subschema Control Block 16-20

SUBSCHEMA-CONTROL 15-25

SUBSTRING built-in function 7-51

SUBTRACT command 13-11

suspense file 15-37

symbol table 3-15

SYSIDMS parameters

 for physical requirements B-24

system fields 21-30, 21-34

system functions 1-9

See also function

 BACKWARD 1-9, A-13

 FORWARD 1-9, A-13

 HELP 1-9, 4-16

 POP 1-9

 POPTOP 1-9

 QUIT 1-9

system functions (*continued*)

 RETURN 1-9

 SIGNOFF 1-9, G-7

 SIGNON 1-9, G-6

 TOP 1-9

system records

See ADSO-APPLICATION-GLOBAL-RECORD

See ADSO-APPLICATION-MENU-RECORD

See ADSO-STAT-DEF-REC

SYSTEM statement (ADSOBSYS) D-38

system-supplied data fields 11-6

 \$ERROR-COUNT 11-9

 \$INPUT-COUNT 11-9

 \$OUTPUT-COUNT 11-9

 CURSOR-COLUMN 11-8

 CURSOR-ROW 11-8

 DATE 11-8

 DB-NAME 11-6

 DIRECT-DBKEY 11-6

 ERROR-STATUS 11-8

 LENGTH 11-8

 NODE-NAME 11-6

 TIME 11-9

T

tables

 diagnostic 3-15

 symbol 3-15

task 4-24

task application table (TAT) 2-39, 4-4, 20-12

See also ADSOBTAT

See also ADSOTATU

task code, for TCF 2-4

Task Codes screen (ADSA) 2-39

TAT

See task application table (TAT)

TAT Update Utility D-58

TCF

See transfer control facility

test condition

See conditional expressions

test conditions

 WHERE clause 16-65

THEN command-statement 14-5

TODAY built-in function 7-54

TOLOWER built-in function 7-55

TOMORROW built-in function 7-56

TOP function 1-9

See also system functions

TOUPPER built-in function 7-57
 TRACE 20-13, H-6
 trace facility H-5—H-6
 trailing sign built-in functions
 GOODTRAILING 7-26
 TRAILING-TO-ZONED 7-58
 ZONED-TO-TRAILING 7-67
 TRAILING-TO-ZONED built-in function 7-58
 transaction statistics
 See dialog statistics
 TRANSFER command 15-34
 status test outcome 8-14
 transfer control facility 2-4, 3-4—3-5
 TRANSLATE built-in function 7-59
 trigonometric built-in functions
 ARCCOSINE-DEGREES 7-12
 ARCCOSINE-RADIANS 7-12
 ARCSINE-DEGREES 7-13
 ARCSINE-RADIANS 7-13
 ARCTAN-DEGREES 7-14
 ARCTAN-RADIANS 7-14
 COSINE-DEGREES 7-16
 COSINE-RADIANS 7-16
 SINE-DEGREES 7-46
 SINE-RADIANS 7-46
 TRUNCATED condition 8-20

U

usage modes 4-26
 exclusive 16-56
 protected 16-56
 retrieval 16-55
 shared 16-56
 update 16-55
 user program
 DC RETURN statement 15-27
 linking 15-26
 user program function
 See program function
 utilities
 ADSOBCOM D-4—D-29, D-36
 ADSOBSYS D-37—D-48
 ADSOBTAT D-48—D-56
 ADSORPTS B-3—B-30
 ADSOTATU D-57—D-59
 utility commands
 ABORT 20-4
 ACCEPT 20-8
 INITIALIZE RECORDS 20-10
 SNAP 20-11

utility commands (*continued*)
 TRACE 20-13
 WRITE PRINTER command 20-14

V

variable dialog block (VDB) 20-12
 variable expression description element
 See VXDE module
 variable terms
 types of 5-8
 variables
 arithmetic expressions 6-3—6-6
 conditional expressions 8-3
 constants 9-3
 data fields 11-3
 entity names 11-12
 error expressions 10-6
 system-supplied 11-6
 target fields 11-10
 user-defined 11-4
 variable target fields 11-10
 VDB
 See variable dialog block (VDB)
 VDE module
 processing of F-17
 vector call codes B-11
 VERIFY built-in function 7-61
 VM/ESA commands
 ADSOBCOM D-33
 ADSOBSYS D-44
 ADSOBTAT D-54
 ADSORPTS B-27
 VM/ESA systems
 See VM/ESA commands
 VSAM data sets
 See native VSAM data sets
 VSE/ESA JCL
 ADSOBCOM D-31
 ADSOBSYS D-42
 ADSOBTAT D-52
 ADSORPTS B-26
 VXDE module F-4, F-8—F-17
 processing of F-17

W

WEEKDAY built-in function 7-62
 WHAT-RECEIVED system field 21-30, 21-34
 WHERE clause 16-65
 comparison expression 16-66

WHERE clause (*continued*)

conditional expression 16-65

test condition 16-66

WHILE command 14-10

with FIND/OBTAIN DB-KEY 16-36

WORDCAP built-in function 7-65

WRITE PRINTER command

WRITE TO LOG/OPERATOR

WRITE TO LOG/OPERATOR command 20-18

WRITE TRANSACTION command 15-36

X

XDE module F-4, F-6, F-8—F-17

Y

YESTERDAY built-in function 7-66

Z

zoned decimal data type 5-13

ZONED-TO-TRAILING built-in function 7-67